SEMANTICS-BASED METRICS AND ALGORITHMS FOR DYNAMIC CONTENT IN WEB DATABASE APPLICATIONS

Stavros Papastavrou

University of Cyprus, 2009

Dynamic Content Technology brings together traditional databases and the Web by allowing for data in databases to be viewed and updated as dynamic web documents. Since the mid 90's, when the Common Gateway Interface emerged, dynamic content technology has facilitated the adaptation of traditional applications to the on-line world. On-line bookstores, virtual communities, web mail, search engines, goods bidding and real-time stock trading are typical examples of such on-line web database applications.

Materialization of dynamic web pages from web databases is a procedure that requires considerable resources, since local or remote databases are accessed and lengthy code is executed to produce the web pages. Consequently, web sites exhibit slow download times when the number of concurrent client sessions increases due to their popularity.

A traditional approach to reduce the time for materializing a dynamic web page, especially under heavy workload, is through caching. Caching allows for parts of dynamic pages, better known as fragments, to be reused from main memory rather than recomputed. In this way, computational resources are spared and Quality of Service (QoS) is enhanced. However, Quality of Data (QoD) may be reduced since fragments with stale/old data could be used. This means that content delivered to web users may contain invalid information, having the same or worse negative impact to slow response times. Thus, a big challenge in the materialization of dynamic web pages has been the trade-off between QoS in terms of response time and QoD with respect to data freshness.

Stavros Papastavrou

This dissertation argues that current approaches for web content materialization that balance QoS and QoD fail to fully capture the characteristics of modern web applications. Further, this dissertation contributes new semantics-based materialization algorithms for web-based dynamic content applications that achieve a better balance between QoS and QoD compared to existing syntactic-based approaches. The novelty of our algorithms lies on the consideration and exploitation of (a) the dependencies among dynamic content fragments and templates and (b) the user request patterns.

Content dependencies are characterized by introducing two new metrics: QoL (Quality of Link) and QoSV (Quality of Set-View). The former measures the ability of the user to navigate between dynamic pages and the latter the set-wise consistency of content fragments inside dynamic pages. The two new metrics substitute the traditional single metric of QoD. In addition, this dissertation exploits the notion of Usage Plans in the context of dynamic web pages. Their purpose is to capture the temporal recurrent behavior of web users toward improving QoD.

Extensive experimental evaluation, based on an on-line Bookstore application, has confirmed the advantages of our semantics-based materialization algorithms compared to the existing syntactic-based ones. With a minimum offline setup, our approach has direct applicability to e-commerce vendors seeking to boost performance with less hardware under higher workloads. Other applications of our approach include content adaptation for mobile devices.

Stavros Papastavrou

Greek Abstract Page #1

Stavros Papastavrou

Greek Abstract Page #2

Stavros Papastavrou

SEMANTICS-BASED METRICS AND ALGORITHMS FOR DYNAMIC CONTENT IN WEB DATABASE APPLICATIONS

Stavros Papastavrou

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Cyprus

Recommended for Acceptance

by the Department of Computer Science

June, 2009

Stavros Papastavrou

© Copyright by

Stavros Papastavrou

All Rights Reserved

2009

Stavros Papastavrou

APPROVAL PAGE

Doctor of Philosophy Dissertation

SEMANTICS-BASED METRICS AND ALGORITHMS FOR DYNAMIC CONTENT IN WEB DATABASE APPLICATIONS

Presented by

Stavros Papastavrou

Research Supervisor _____

Committee Member _____

Committee Member _____

Committee Member _____

Committee Member _____

University of Cyprus

June, 2009

Stavros Papastavrou

ACKNOWLEDGEMENTS

Many thanks to my professors and advisors who inspired and supported me at the University of Cyprus and the University of Pittsburgh. Special thanks to Dr. Panos K. Chrysanthis who did not give up on me, even at very difficult times. Special thanks to Dr. George Samaras who introduced and kept me into research. Special thanks to Dr. Paraskevas Evripidou who inspired me in concurrent programming, on which my experimental implementation is based. Special thanks to Dr. Evaggelia Pitoura and Dr. Alexandros Labrinidis whose research and comments triggered my dissertation. Also, I would like to thank them for their feedback and suggestions on my dissertation. Special thanks to Dr. Marios Dikaiakos for his insightful and helpful comments on my dissertation. Finally, a warm thank you to Savoulla Efstathiou for her endless guidance on administrative issues.

Stavros Papastavrou

To my wife Stella and daughter Antri

Stavros Papastavrou

ΣΗΜΑΣΙΟΛΟΓΙΚΑ ΜΕΤΡΑ ΚΑΙ ΑΛΓΟΡΙΘΜΟΙ ΓΙΑ ΔΥΝΑΜΙΚΟ ΠΕΡΙΕΧΟΜΕΝΟ ΣΕ ΕΦΑΡΜΟΓΕΣ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ ΣΤΟ ΠΑΓΚΟΣΜΙΟ ΠΛΕΓΜΑ ΠΛΗΡΟΦΟΡΙΩΝ

ΠΕΡΙΛΗΨΗ

Η Τεχνολογία Δυναμικού Περιεχομένου (Dynamic Web Content) συσχετίζει τις Παραδοσιακές Βάσεις Δεδομένων (Traditional Databases) με το Παγκόσμιο Πλέγμα Πληροφοριών (World Wide Web), επιτρέποντας την επισκόπηση και ενημέρωση των βάσεων δεδομένων μέσω δυναμικών ιστοσελίδων. Με την εμφάνιση του Common Gateway Interface (CGI), η τεχνολογία δυναμικού περιεχομένου έχει υποβοηθήσει στην μεταφορά των παραδοσιακών δημοφιλών εφαρμογών στο διαδικτυακό κόσμο. Παραδείγματα δημοφιλών εφαρμογών αποτελούν τα διαδικτυακά βιβλιοπωλεία, οι εικονικές κοινότητες, οι μηχανές αναζήτησης, οι δημοπρασίες αγαθών, το διαδικτυακό ηλεκτρονικό ταχυδρομείο και, τέλος, οι χρηματιστηριακές πλατφόρμες.

Η υλοποίηση δυναμικών ιστοσελίδων από βάσεις δεδομένων είναι μια διαδικασία που απαιτεί σημαντικούς υπολογιστικούς πόρους, εφόσον εμπλέκει πρόσβαση σε τοπικές ή κατανεμημένες βάσεις δεδομένων, καθώς και την εκτέλεση μακροσκελούς κώδικα. Ως αποτέλεσμα, ο χρόνος απόκρισης δημοφιλών δυναμικών ιστοσελίδων αυξάνεται, όταν ο αριθμός των συνδεδεμένων χρηστών ανεβαίνει.

Η παραδοσιακή προσέγγιση για μείωση του χρόνου εκτέλεσης της υλοποίησης μιας δυναμικής ιστοσελίδας, ειδικά όταν ο αριθμός των χρηστών είναι αυξημένος, επιτυγχάνεται διαμέσου της Εναποθήκευσης (Caching). Η εναποθήκευση επιτρέπει την επαναχρησιμοποίηση κομματιών περιεχομένου (content fragments) μιας δυναμικής ιστοσελίδας από Συστήματα Εναποθήκευσης. Με αυτή την πρακτική διασώζονται πολύτιμοι υπολογιστικοί πόροι και η Ποιότητα Εξυπηρέτησης (QoS) αναβαθμίζεται. Ωστόσο, η Ποιότητα Δεδομένων (QoD) ίσως μειώνεται, όταν τα εναποθηκευμένα μέρη ιστοσελίδων που χρησιμοποιούνται δεν είναι πρόσφατα ενημερωμένα. Συνεπώς, το περιεχόμενο που παραδίδεται στους χρήστες πιθανώς να περιέχει άκυρες πληροφορίες, επιφέροντας δυσχερέστερες επιπτώσεις και από τους αργούς χρόνους απόκρισης. Ως αποτέλεσμα, η μεγάλη πρόκληση στην υλοποίηση δυναμικών ιστοσελίδων συνίσταται στη επίτευξη συμβιβαστικής ισορροπίας μεταξύ της ποιότητας εξυπηρέτησης, υπό μορφή χρόνων απόκρισης, και της ποιότητας δεδομένων, υπό μορφή έγκαιρης ενημέρωσης των Δεδομένων (Data Freshness).

Η παρούσα Διατριβή αμφισβητεί την αποτελεσματικότητα των τρεχόντων προσεγγίσεων συμβιβαστικής ισορροπίας μεταξύ ποιότητας εξυπηρέτησης και ποιότητας δεδομένων, εφόσον αυτές

Stavros Papastavrou

αποτυγχάνουν να ενσωματώσουν τα χαρακτηριστικά των δημοφιλών διαδικτυακών εφαρμογών. Επιπρόσθετα, η παρούσα Διατριβή συνεισφέρει καινοτόμους αλγόριθμους υλοποίησης δυναμικών ιστοσελίδων για διαδικτυακές εφαρμογές, οι οποίοι βελτιώνουν την ισορροπία μεταξύ ποιότητας εξυπηρέτησης και ποιότητας δεδομένων σε σχέση με τις τρέχουσες προσεγγίσεις. Η καινοτομία των αλγορίθμων έγκειται στην ενσωμάτωση και εκμετάλλευση (α) των διασυνδέσεων και εξαρτήσεων μεταξύ μερών περιεχομένου δυναμικών ιστοσελίδων, και (β) των μοτίβων πρόσβασης ιστοσελίδων των χρηστών.

Οι διασυνδέσεις και εξαρτήσεις των μερών περιεχομένου χαρακτηρίζονται από την εισαγωγή δύο νέων μέτρων: της Ποιότητας Διασύνδεσης (QoL) και της Ποιότητας Δια-Προεπισκόπησης (QoSV). Το πρώτο μέτρο ποσοτικοποιεί τη δυνατότητα του χρήστη να πλοηγείται μεταξύ δυναμικών ιστοσελίδων, ενώ το δεύτερο μέτρο ποσοτικοποιεί την ικανοποίηση των συσχετίσεων μεταξύ των διασυνδεδεμένων μερών περιεχομένου σε μία δυναμική ιστοσελίδα. Τα δύο αυτά μέτρα αντικαθιστούν το παραδοσιακό μέτρο της ποιότητας δεδομένων. Επιπρόσθετα, η παρούσα Διατριβή εισάγει την έννοια των Πλάνων Χρήσης (Usage Plans), τα οποία χαρτογραφούν την επαναλαμβανόμενη συμπεριφορά των χρηστών, με σκοπό την βελτίωση της ποιότητας δεδομένων.

Εκτεταμένα πειράματα, με τη χρήση εφαρμογής διαδικτυακού βιβλιοπωλείου, έχουν επιβεβαιώσει τα πλεονεκτήματα των προτεινόμενων αλγορίθμων υλοποίησης, έναντι των παραδοσιακών προσεγγίσεων. Με ελάχιστο κόστος εγκατάστασης, η παρούσα προσέγγιση μπορεί να εφαρμοστεί σε υπάρχοντα συστήματα δυναμικών εφαρμογών διαδικτύου, αποδίδοντας αυξημένο όγκο διεκπεραίωσης εργασιών, με την υποστήριξη αυξημένου αριθμό τρεχόντων χρηστών. Προτείνονται θεωρητικές και πρακτικές εφαρμογές των ερευνητικών αποτελεσμάτων, που εστιάζονται, ανάμεσα σε άλλα, στην προσαρμογή περιεχομένου σε κινητές συσκευές.

Stavros Papastavrou

TABLE OF CONTENTS

Stavros Papastavrou

Stavros Papastavrou

Stavros Papastavrou

Stavros Papastavrou

Stavros Papastavrou

LIST OF TABLES

Stavros Papastavrou

LIST OF FIGURES

Stavros Papastavrou

Stavros Papastavrou

Stavros Papastavrou

Stavros Papastavrou

Stavros Papastavrou

Stavros Papastavrou

Chapter 1

Introduction

1.1 Motivation

Over the past decade, *dynamic content technology* has transformed the World Wide Web [22] from a vast static document repository to a collection of numerous on-line Web database applications varying from e-marketplaces, search engines, Web mail and, lately, virtual worlds. Instead of using static HTML documents [21], Web database applications are driven by dynamically generated HTML documents, widely known as *dynamic Web pages* [23], from data in databases. Such pages capture the business logic of the application and provide the means for users to navigate and interact with the application though URL links and HTML Forms [20].

According to the typical system architecture (Figure 1), user requests for dynamic Web pages are forwarded through proxy servers to the Web server, which is the public entry point for a Web database application. From there, they are routed to an application server (content middleware), which generates content by querying application databases. In brief, dynamic Web pages are syntactically defined with the use of static template files, containing simple HTML tags that mark out the layout of the dynamic parts of the pages, widely known as *dynamic content fragments*.

Stavros Papastavrou

Figure 2 shows a result page from the popular Google search engine, which generates dynamic Web pages [30]. The figure focuses on the top three fragments which are indicated with dotted boxes. The first fragment from top contains the HTML Form that provides the user with the means of entering a search phrase. The second fragment displays statistics on the results of the previous search, such as the number of relevant results, the time required for executing the search. The third fragment contains the list of the first ten results, as well as links for navigating to the corresponding Web page for every result in that list.

Similarly, Figure 3 focuses on the main three fragments from a book information page of the popular Amazon bookstore. The main fragment displays the book information. The fragment on the right contains links for adding the book into the shopping cart of the user and the fragment at the bottom lists a relate book.

The application server materializes a fragment by processing a specific block of script code. An example of a content fragment using the ColdFusion [107] scripting language is presented in Figure 4. In the example, the fragment executes a query on the application database and then formats the query results with HTML.

The purpose of the proxy server in the typical system architecture, is to store materialized content for future reuse. This practice is widely known as *Caching* [9, 27] and is a popular approach for sparing processing resources at the server side, as well as for reducing network traffic. In fact, caching is an extremely effective technique in typical computer systems and applications whose access patterns have locality of reference, that is data accessed close together in time. For example, caching is employed on CPUs, hard drives, Domain Name Servers, database servers and user browsers.

But caching may compromise the quality of data (QoD) of delivered content in terms of freshness if it contains stale/old data. Hence, dynamic generation of content fragments in the presence

Stavros Papastavrou

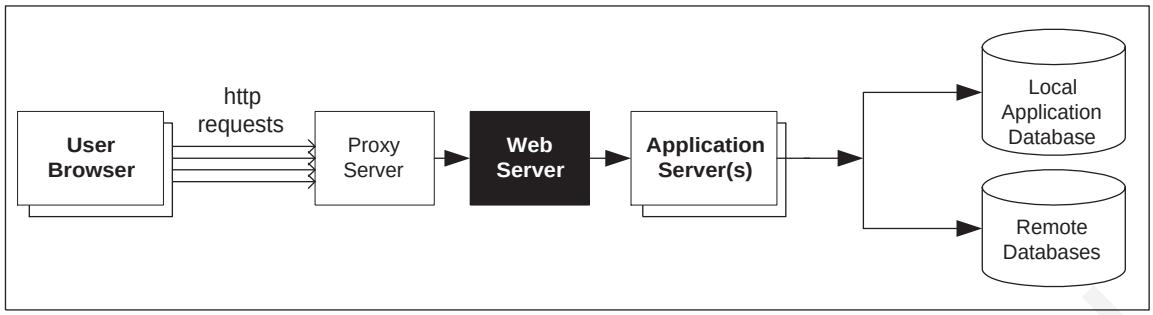

Figure 1: Typical System Architecture for Generating Dynamic Web Pages for Web Database Applications

of caching requires special handling/attention to ensure the success of a web database application. This dissertation focuses on this problem and in particular on the trade-off of QoS in terms of response time and QoD in terms of freshness.

1.2 Classification of Dynamic Web Database applications

Dynamic content technology has enabled a wide range of dynamic Web database applications with various semantics and purposes. To put the contributions of this dissertation into context, we classify the applications along three characteristics: *Target Group*, *Database Update Rate* and *Content Initiator*. This classification is illustrated in Figure 5, which shows the three characteristics as the axes of a 3-dimensional *application comparative structure*.

We explain in brief the three characteristics for Web database applications:

- **Target Group**. It refers to the popularity granularity of content and affects the ability of sharing materialized content across users.

  Broad-interest applications include on-line news, football results and stock market information. They serve dynamic pages with predefined structure and content and target a vast number of users. Content is heavily reused and is stored on various locations such as proxy servers and Content Distribution Networks (CDN) [57].

Stavros Papastavrou

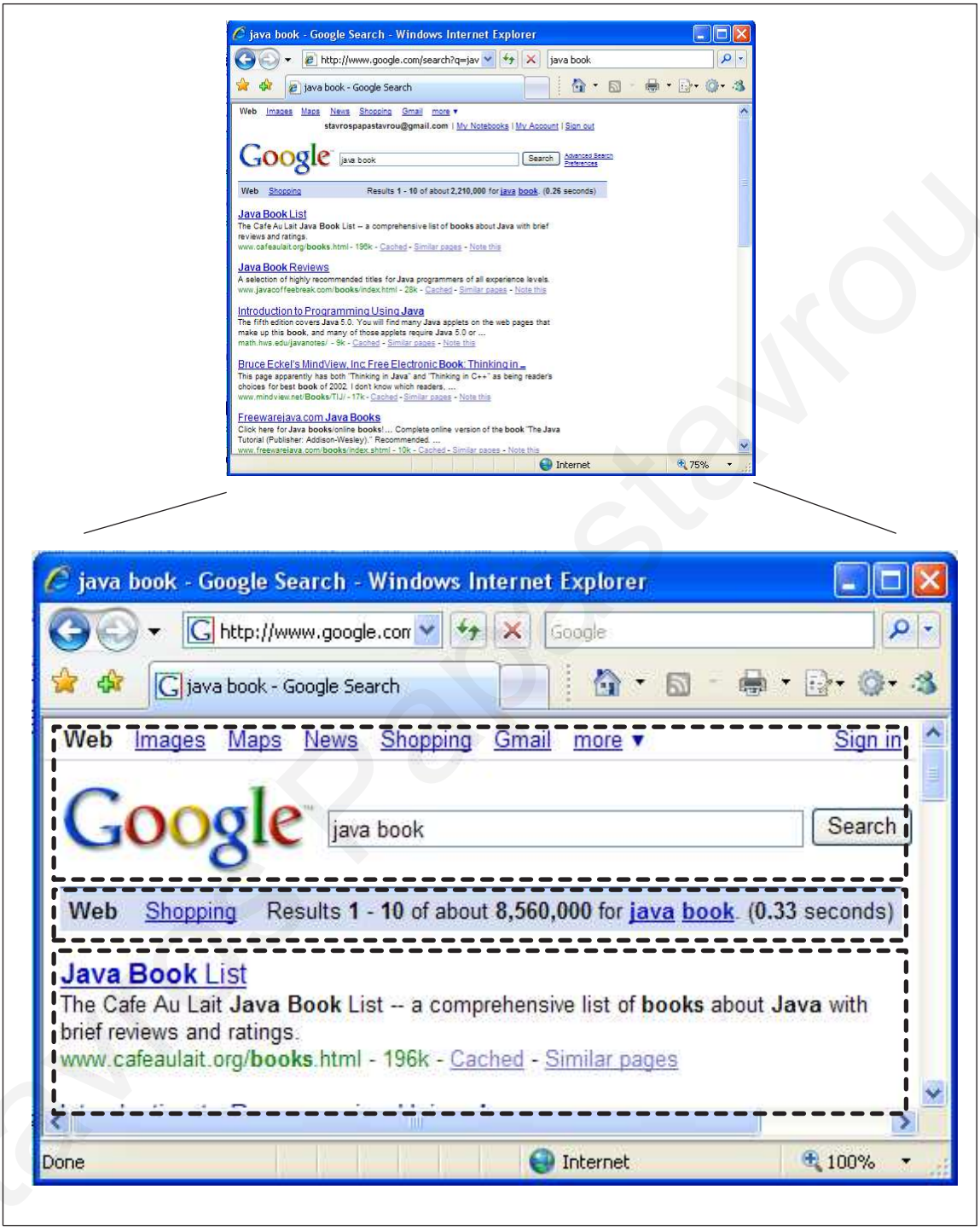

Figure 2: The Search Results Dynamic Web Page of the Google Search Engine. The Figure focuses on the top three fragments of the page.

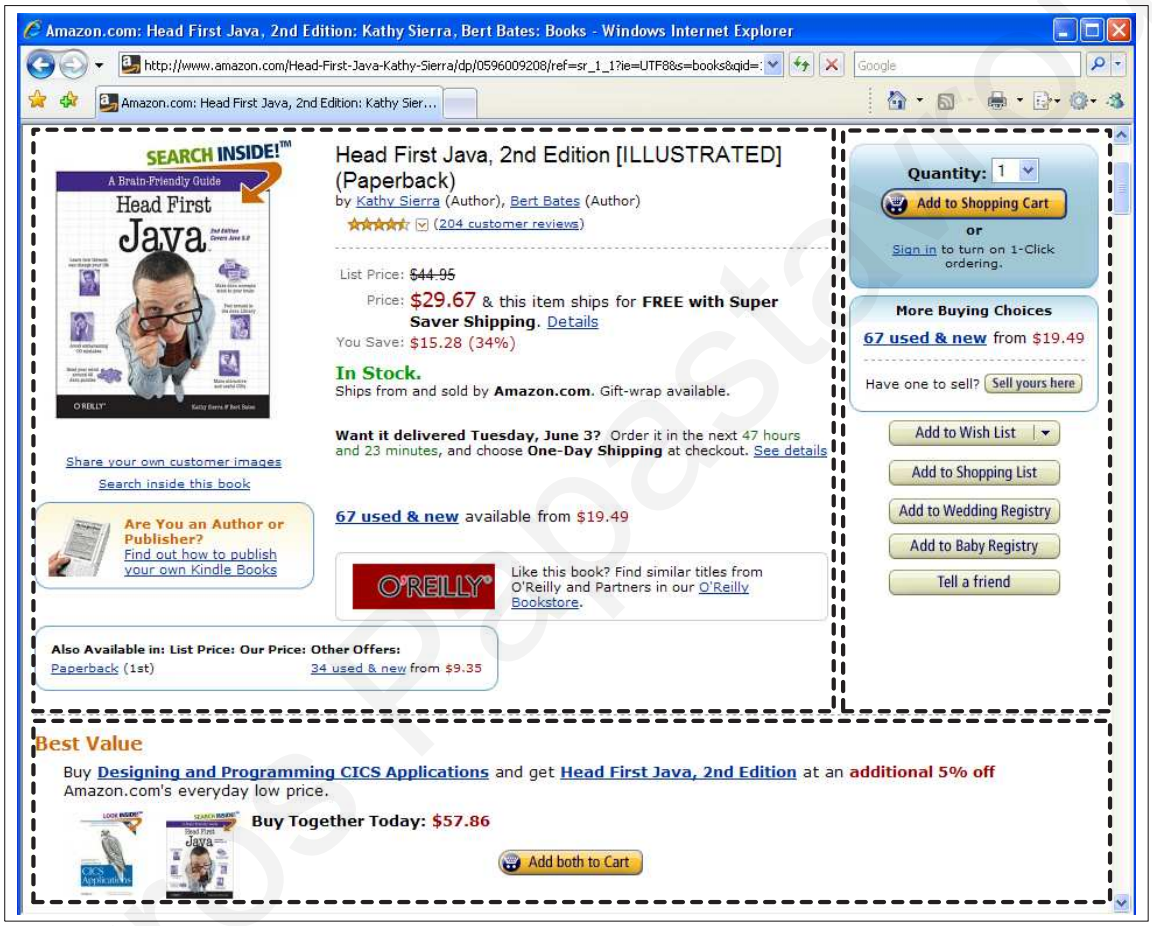

Figure 3: A Book Information Page from the Amazon Website. The Figure focuses on the main three fragments of the page.

Stavros Papastavrou

```
// BEGIN FRAGMENT
<cfquery name="book_search"
  datasource="appDATABASE">
  SELECT   *
  FROM books
  WHER title LIKE "Java Book">
</cfquery>

<p>
The books found:
<cfoutput>#book_search.RecordCount#</cfoutput>
<p>

List  of books found<p>
<table border=1>
<tr><td> Title  </td> <td> Price </td>

<cfloop query="employeesOver40">
<tr>
  <td>  <cfoutput>#book_search.title#</cfoutput> </td>
  <td>  $<cfoutput>#book_search.price#</cfoutput> </td>
</tr>
</cfloop>
</table>
//END FRAGMENT
```

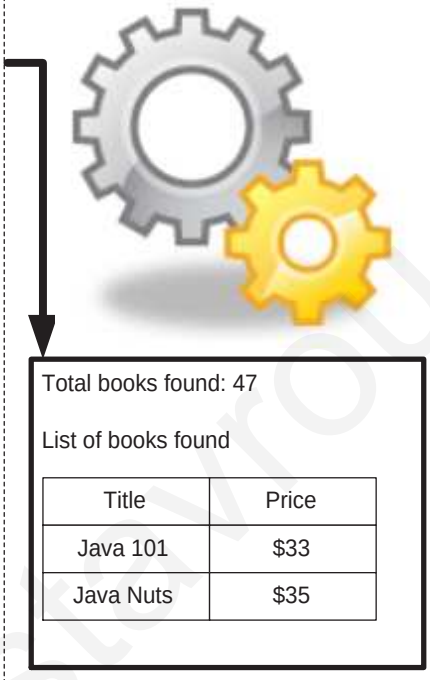

Figure 4: Source Code for a Content Fragment

On the other hand, an example of a narrow-interest application is a product customization on-line store. Narrow-interest applications serve dynamic pages with user-personalized content that addresses the specific needs and expectations of single users. The combination of a user's choices (i.e., PC components customization) and the overhead of deciding whether content fragments are reusable by other users, prohibits content reuse across users.

- **Database Update Rate**. It refers to the frequency of changes to the application database objects involved in content materialization and affects the chances of content reuse.

  A PC customization application may not change its products for several weeks, thus leaving the corresponding objects in the application database unmodified for a long period. In this way, some dynamic Web pages of the application, such as a catalog or the basic specifications of a PC, can be served from a cache.

Stavros Papastavrou

Popular on-line news applications have a moderate frequency of updates (i.e., hourly or daily) and, therefore, Web pages are not materialized on every request. Subsequently, materialized pages can be sent on a regular basis to proxies and CDNs for immediate reuse. On the other hand, a stock-related application has very frequent updates on its database and requires more frequent content materialization in order to deliver current information.

- **Content Initiator**. It refers to the side that triggers the materialization of content.

  Data-driven applications, such as stock-related sites and on-line news, can perform background materialization of content and caching in response to incoming data updates. On the other hand, user-driven applications, such as product search and customization, require that content be materialized on user request.

According to their semantics, Web database applications borrow from more than one of the above three characteristics. Figure 6 displays a rough classification of popular modern applications on the application classification structure. We discuss below some interesting examples:

For e-commerce applications (top left side of the application comparative structure), such as an on-line bookstore and product customization, content materialization is mostly triggered by user requests. Application data does not change very often and the materialized content targets the expectations of single users.

For applications based on the on-line transaction processing model (OLTP) (bottom right side), an enterprize-level repository designed to facilitate reporting and analysis [50], data is updated frequently and content materialization is data-driven. Applications such as on-line stock information Websites fall into this category.

On-line bidding applications (two front edges of the application comparative structure), such as the popular ebay.com and ubid.com, share characteristics of both e-commerce and data-warehouse

Stavros Papastavrou

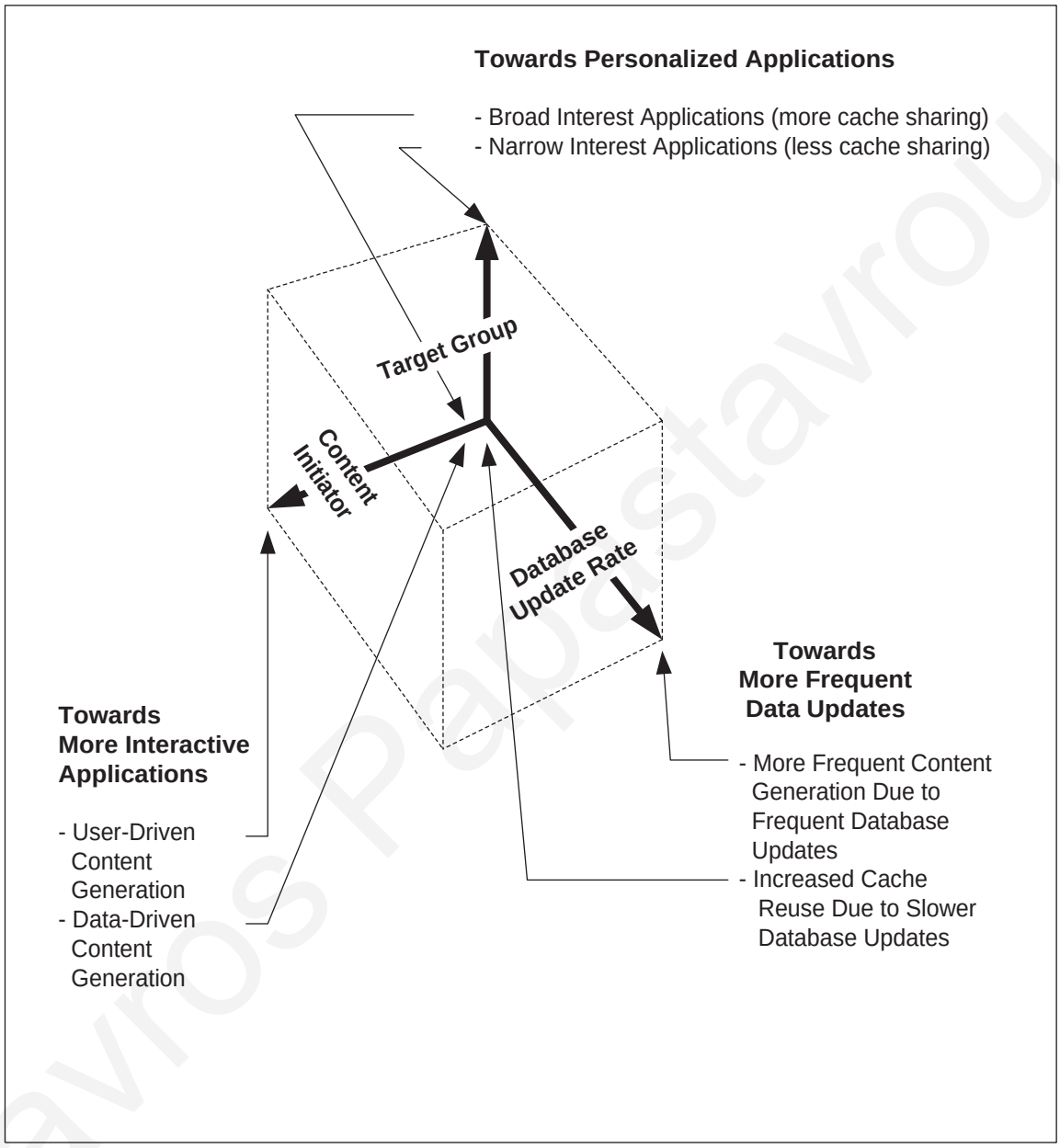

Figure 5: The Application Comparative Structure for Web Database Applications. The three dimensions represent the three characteristics of Web database applications along with explanation on their semantics.

Stavros Papastavrou

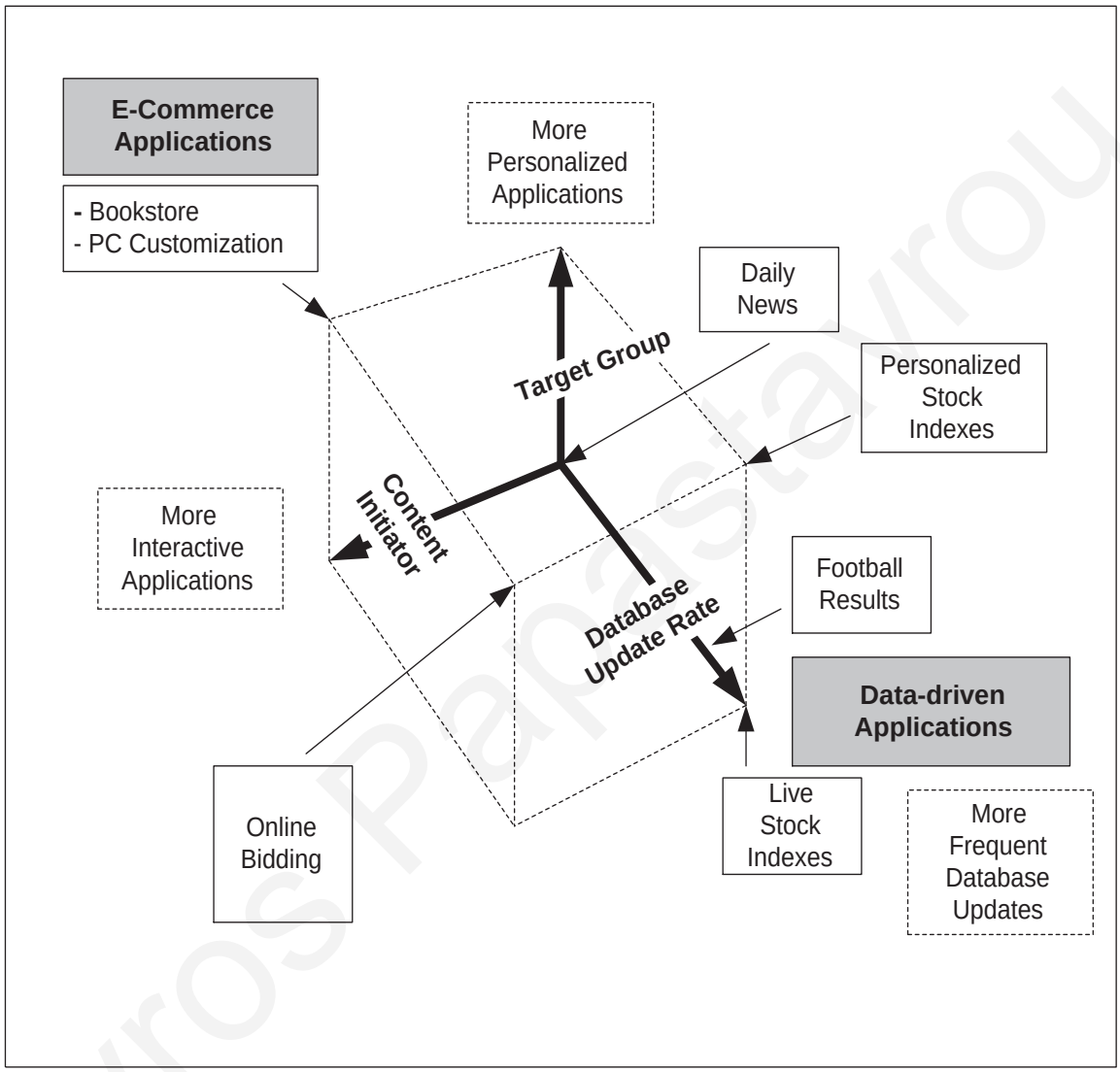

Figure 6: A Rough Classification of Web Database Applications Based on the three Characteristics of Target Group, Content Initiator and Database Update Rate. Interactive e-commerce applications are classified on the top left side. Data-driven applications are classified on the bottom right side. At the far right side, an on-line bidding application shares the semantics of both e-commerce and data-driven applications.

Stavros Papastavrou

applications. This is because, content materialization is both user and data-driven, in the sense that updates on the database are triggered by the users themselves on every bidding.

1.3 Current State of Art in Dynamic Web Pages

Materialization of dynamic Web pages from Web databases is a procedure that requires considerable resources, since local or remote databases are accessed and lengthy code is executed to produce HTML [11]. For example, the product customization page for the www.higrade.com computer retailer Web site contains approximately 2000 lines of script code with more than ten database requests distributed across eight content fragments. The materialized content in this application contains complex Web forms for product customization with price option adjustments.

Subsequently, Web sites exhibit slow download times, especially when the number of concurrent user sessions rises. For data-intensive Web database applications, such as e-commerce, the database side is identified as the potential performance bottleneck [11]. Reportedly, more than 20 billion dollars are lost every year due to excessive delays in e-commerce Web pages that force users to abandon their session [83]. For instance, it is estimated that slow Web sites resulted in almost one billion dollars in lost revenue, during the period between Thanksgiving and New Year's Eve in 2006.

Since the mid 90's, early dynamic content technology focused on improving Quality of Service (QoS) in terms of user and server-perceived response times, by refining the implementation of the three principles of content *Caching*, *Fragmentation* and *Polymorphism* [92]. More specifically,

- Fragmentation is the practice of dissecting dynamic Web pages into content areas using template files. The content areas, widely known as content fragments, can be materialized separately [34, 90]. Subsequently. fragmentation benefits QoS by providing support for materializing, on-demand, only specific parts of a dynamic Web page.

Stavros Papastavrou

- According to the principle of caching, materialized content fragments may be independently cached on the server-side [7, 40, 41, 52, 55, 104, 114], on proxy servers [2, 74, 99, 104] or even at the user [98]. Caching benefits QoS since computational resources required for materializing a fragment, which is to be reused from cache at a given moment, are spared.

- According to the principle of polymorphism [42], materialized content fragments are assembled together and delivered to the user in various arrangements, therefore adding an extra low-cost layer of dynamism. Polymorphism enhances QoS since it supports the dynamic arrangement of fragments without the need to re-materialize content. Polymorphism can be implemented on the server side, the proxy side, or even at the user side.

The employment of the principles of caching, fragmentation and polymorphism to improve QoS, however, compromises the QoD of delivered content in terms of data freshness. That is, the degree to which materialized content relates to the underlying application database objects that were queried during content materialization [10, 36, 47, 53, 103, 110].

QoD is as well important as QoS, since fresh content guarantees that no inconsistent or stall information is sent to Web users. Content that is not fresh could potentially cause problems to a user session since it may contain outdated/invalid URL links and HTML Forms.

Subsequently, a big challenge in this context has been the trade-off between QoS in terms of response time and QoD with respect to data freshness. There have been several attempts to balance QoS with QoD, through related research, in order to maintain acceptable user response times and overall quality of delivered content [18, 29, 32, 37, 38, 43, 66, 71, 97].

In brief, current QoS-QoD balancing approaches allow for content fragments to be reused from cache, under heavy workload. In this way, computational resources are spared and QoS is enhanced at the expense of QoD, since stale/old data are used. More specifically, the basic

Stavros Papastavrou

principle behind those proposals is that a variable number of "less important to the users" content fragments, per dynamic Web page, are not materialized on every user request. Instead, those (possibly stale) fragments are reused from the cache, sparing resources for promoting QoS. This user-wise preference on the importance of individual fragments is facilitated by *user profiling*, a popular methodology that enables the users of a Web application to define their specific needs and expectations. User profiling is driven by session and user tracking techniques such as Web cookies [59] and session-specific variables in application servers.

1.4 Problem Statement

In the present dissertation, we take the position that current QoS-QoD balancing proposals are not suitable for modern Web database applications, such as e-commerce Web applications. We argue that the QoD is used as a syntactic metric that does not consider a number of characteristics that emerge from the conceptual design of the Web database applications. Those characteristics are potential sources of problems (summarized in Table 1). More specifically,

Fragment Dependencies The relations among content fragments inside a template, referred to as *Fragment Dependencies*, which if left unsatisfied may cause the following two problems: (a) prevent specific fragments in a dynamic page from materializing, in case where dependent fragments use as input data computed by other fragments and (b) serve pairs of highly related fragments to the users containing inconsistent content.

- An example of the former dependency, called *Generation Dependency*, is presented in Figure 7 and relates to the Google search page displayed in Figure 2. In the example given, the dynamic Web page `searchResults.dyn` shows the results of a user-submitted request. The second fragment generates and stores the search results in

Stavros Papastavrou

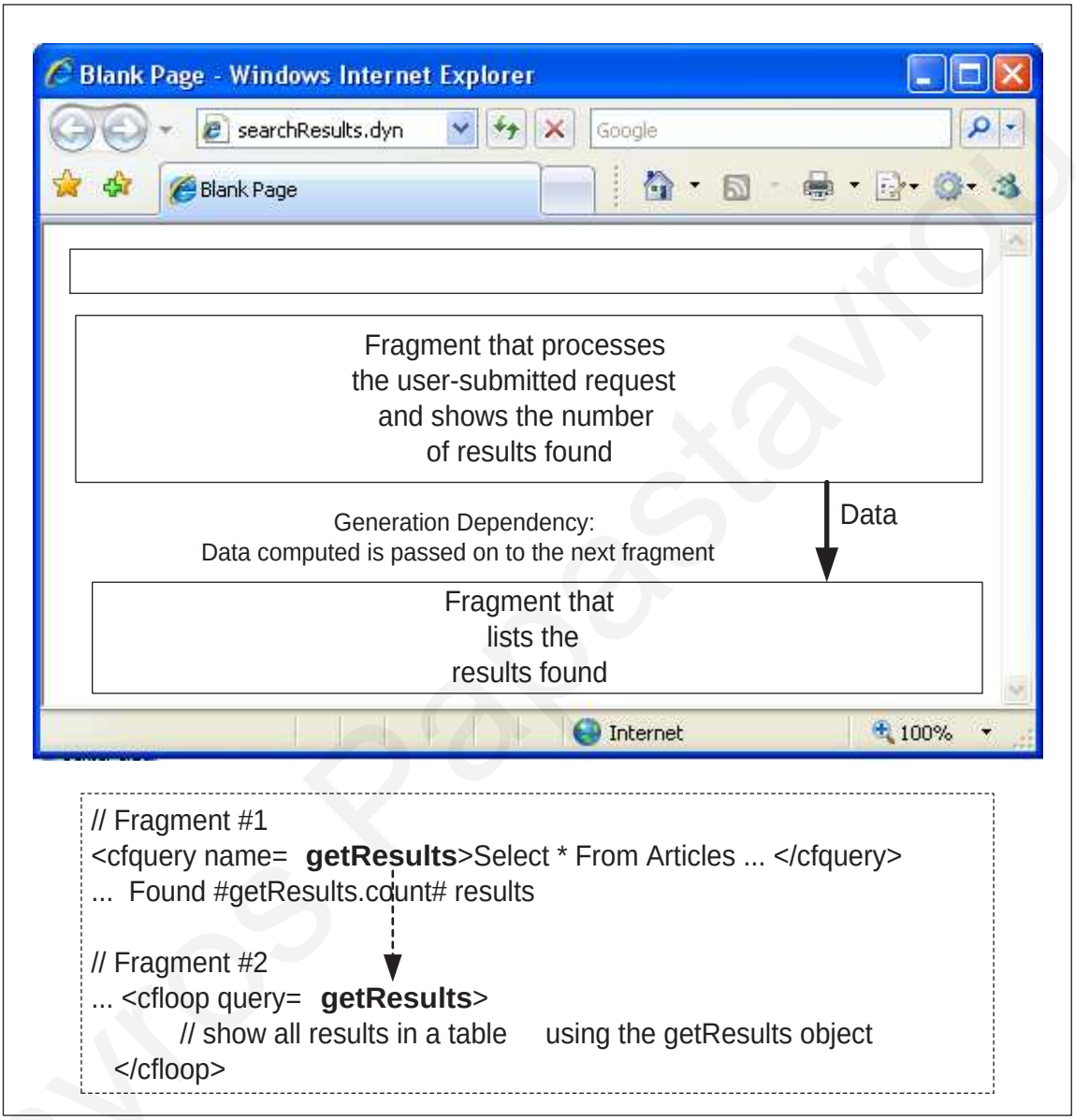

Figure 7: Example of a Generation Dependency using ColdFusion Script that relates to the Search Results Page of Google. The second fragment generates data needed by the third fragment for processing. In this case, the data generated are the results of a user-submitted search. Generation Dependencies emerge when data variables are reused by more than one fragment in a template.

Stavros Papastavrou

a “getResults” data variable and then shows the number of results along with related statistics. The second fragment presents the results in a top-down fashion using the “getResults” data variable. The problem emerges when the first fragment is not materialized by the QoS-QoD balancing algorithm. As a consequence, the second fragment cannot generate the list with the search results, since the “getResults” variable would be null.

Generation dependencies are introduced by the Web developers themselves due to the common practice of reusing early-computed results across later parts of a program. In our context, a data variable that is usually stored as a session variable, is present in the script code of more than one fragments.

- An example of the latter dependency, called *Set-View Dependency*, is presented in Figure 8 and relates to the Amazon book information page displayed in Figure 3. In the example given, the first fragment of the dynamic page `viewBook.dyn` contains the details on the selected book “Head First Java”. The fragment at the bottom shows a suggested book to “Java Nuts”. The problem with set-view dependencies is that no QoS-QoD balancing approach has related provision so that both fragments present consistent information at all times. Subsequently, if the fragment at the bottom is reused from cache, then it would display irrelevant books to the one being viewed. the Set-view dependency between two fragments is satisfied when both dependent fragments are served fresh or reused from cache with the same timestamp.

  Set-View dependencies emerge since the dependent fragments contain data from database tables that are related. In the example of Figure 8, the “Book” table that holds the details for book “Java Nuts” has an one-to-many relation with the “Related Book” table,

Stavros Papastavrou

which holds data on couples of books that are related, according to the application semantics.

Link Dependencies The linking/navigation dependency, referred to as *Link Dependency*, between a content fragment of one template and another template. Figure 9 shows an example of a link dependency that also related to the Amazon book information page displayed in Figure 3. In the example given, the fragment on the right of the dynamic page `viewBook.dyn` contains a link to the dynamic page `shoppingCart.dyn`, so that the user can add the book "Head First Java" in the shopping cart. The problem with link dependencies is that, a fragment that is served from cache by the QoS-QoD balancing algorithm and has link dependencies to the next page that a user will request, can stall the user session since those links are outdated/invalid. In the example given, if reused from cache, the fragment on the right would contain a link for adding an irrelevant book into the shopping cart.

A link dependency between a fragment and a template is introduced by the Web developer with the definition of the URL links from inside the linking fragment to the targeting dependent template. However, this dependency is indirectly based on the relations between the application database tables, since the linking fragment and the dependent template contain content from related database tables. In the example of Figure 9, the linking fragment contains data from the "shoppingCart" table, which has a relationship with the "Book" table, whose content is contained in the dependent template.

Access Patterns The workload characterization or access patterns of users, that is, the popularity and the sequence in which templates are requested. Access patterns have been used in the past in static Web pages to improve QoS and in the context of dynamic Web pages

Stavros Papastavrou

Source of Problem	Introduced by	Problem / Effect
Generation Dependencies	The Web Developer	Prevents Fragments from Materializing
Set-View Dependencies	Relations between Application DB Tables	Related Fragments Present Inconsistent Information
Link Dependencies	Both the Above	No Valid Links to the next Template
User Access Patterns	User Interaction Workflow based on the Conceptual Design of the Web Application	Same as Above

Table 1: Summary of Problems for Current QoS-QoD Balancing Approaches

for predicting and caching in advance (prefetching). However, these were not used for enhancing QoD.

The implications of not considering the workload characterization have a negative impact both on data quality as well as on the overall user experience, since the temporal locality in the request patterns of users is highly related to link dependencies.

More specifically, the problem is that current QoS-QoD balancing approaches do not consider individual pages to be part of a user session in an interactive Web database application. As a result, there is no related provision that prefers fragments for materialization with link dependencies to the next template in the users request sequence. As a consequence, users are not served with fresh links to their next template. The problem with the user access patterns emerges from the user interaction workflow of the Web database application, that is explicitly defined by the Web developer.

1.5 Contributions

The contributions of the present dissertation are both conceptual and practical and are applicable to dynamic Web content applications that fall primarily into the category of user-driven

Stavros Papastavrou

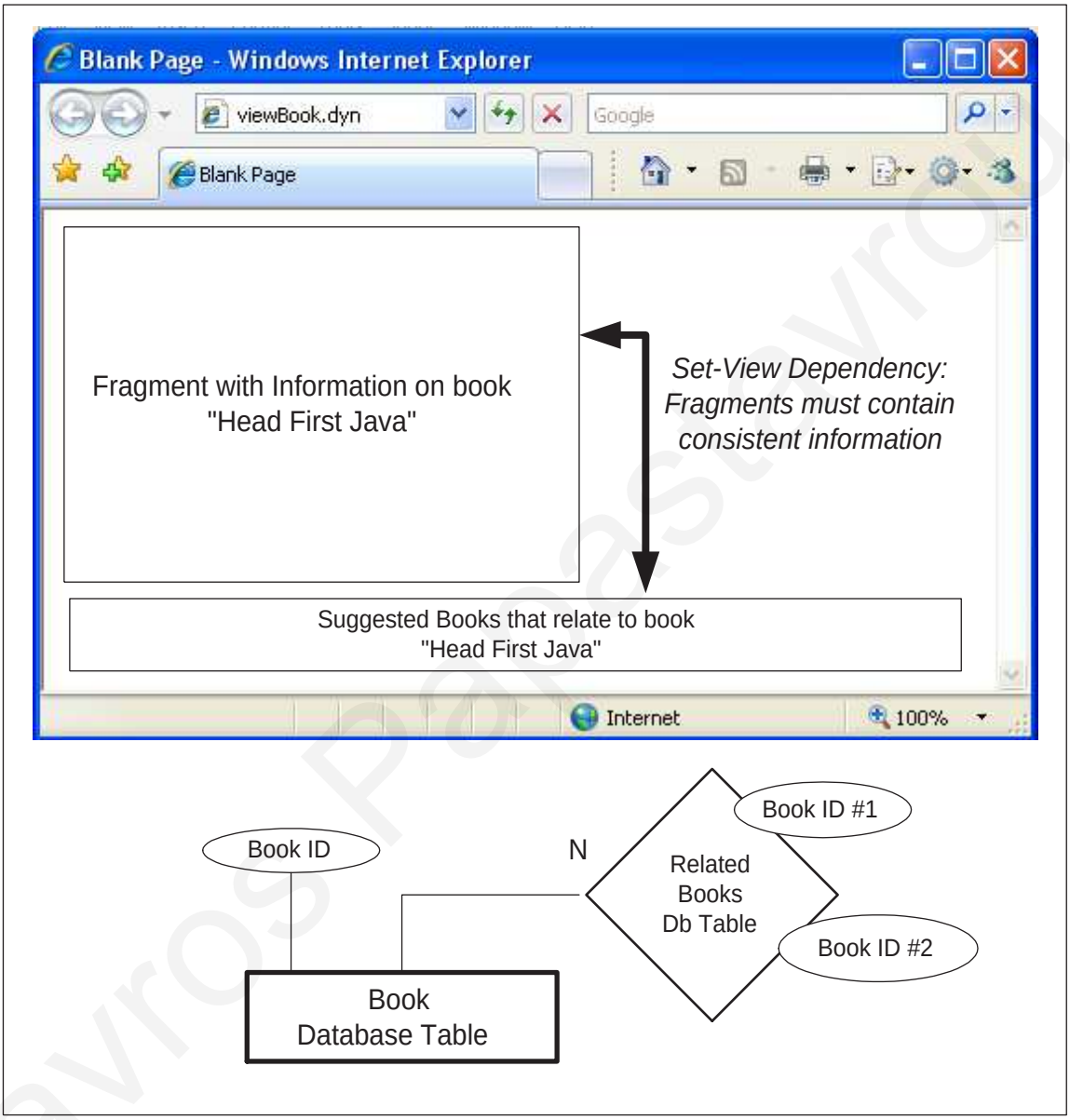

Figure 8: Example of a Set-View Dependency that relates to the Book Information Page of Amazon. Set-view dependency between two fragments emerges since the dependent fragments use data from database tables that are related.

Stavros Papastavrou

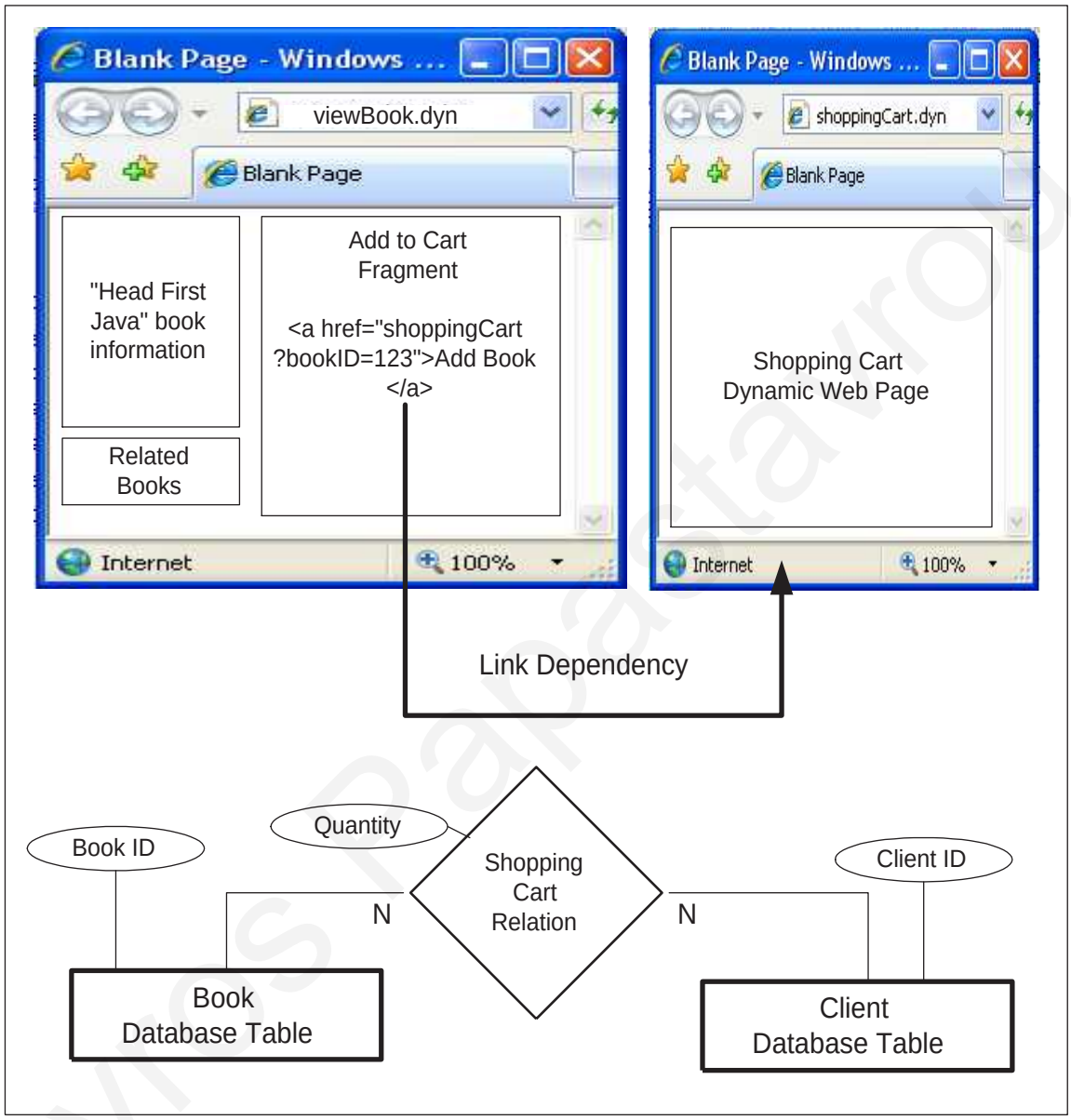

Figure 9: Example of a Link Dependency that relates to the Book Information Page of Amazon. A fragment in the page on the left contains link(s) to the page on the right. A link dependency is explicitly introduced by the Web developer with the definition of the URL links from inside the linking fragment to the targeting dependent template. It also emerges from the relations between the application database tables, since content from related tables is found in both the linking fragment and the target template.

Stavros Papastavrou

applications, as discussed in Section 1.2 and Figure 6. Under certain conditions as we will discuss in the last chapter, our contributions are applicable to other classes of applications as well. Specifically, our key contributions are:

- Content dependencies are identified and classified into *fragment dependencies* and *link dependencies*. Furthermore, fragment dependencies are distinguished into *generation dependencies* and *set-view dependencies* that capture control and data dependencies among fragments within a template, respectively.

- *Quality of Link* (QoL) is introduced as the metric that characterizes link dependencies. As opposed to currently defined QoD, QoL is a semantic metric that measures data freshness in conjunction to the "ability" of a user to navigate from one template to another, given the set of link dependencies between fragments and templates.

- *Quality of Set-View* (QoSV) is introduced as the metric that characterizes set-view dependencies. QoSV is a semantic metric that measures the overall set-wise freshness of fragments inside a template, according to the template's set of set-view dependencies.

- The notion of *Usage Plans* is introduced in the context of dynamic Web pages in order to facilitate user access patterns by exploiting temporal locality of requests. Usage plans are a key component for speculating on the user behavior and are facilitated by a user profiling mechanism.

- We propose two algorithms that select fragments for materialization in respond to user requests by considering (a) the setup of an application in terms of content dependencies, (b) the expected user access patterns of the application and (c) by constantly monitoring system performance and user behavior.

Stavros Papastavrou

The proposed algorithms work in conjunction with a QoS-centric Control Scheme, in order to balance QoS with QoL and QoSV. As opposed to traditional QoD-based approaches, the proposed algorithms handle better the semantic requirements that are introduced by the conceptual design of Web database applications.

1.6 Experiments and Outcome

For the purposes of this dissertation, we have implemented an experimental platform on which we run a real-world on-line bookstore application. An on-line bookstore is a typical e-commerce application, whose dynamic Web pages are fragmented in such a way so that all three content dependency types exists between their fragments and templates.

In addition, our choice for a bookstore application was based on the findings that 41% of Internet users have purchased books [6], according to Nielsen On-line. Through this platform, the proposed algorithms are evaluated and are compared against the traditional QoS-QoD balancing approaches. Extensive tests have been performed using synthetic workload obeying well-established principles, according to the transactional Web e-Commerce benchmark (TPC-W) [78]. Throughout these tests, unexpected user behavior was also considered.

Our evaluation has shown the superiority of the proposed materialization algorithms over the existing approaches. Our algorithms outperform the traditional QoS-QoD approaches by enhancing QoD in terms of QoL and QoSV, thus achieving higher throughput and supporting more concurrent user sessions. Finally, the experiments have identified the interrelationship and the trade-off between QoS, QoL and QoSV.

With a minimum offline setup, our approach has direct applicability to e-commerce vendors seeking to boost performance with less hardware under higher workloads. Our approach can also facilitate other applications such as adapting content for mobile devices and user profiling.

Stavros Papastavrou

1.7 Road Map

The present dissertation is composed by seven chapters:

Chapter 2 contains a comprehensive review on research and technology, focused on improving QoS for Web database applications along the principles of caching, fragmentation and polymorphism. A detailed study on the available QoD metrics is also presented.

Chapter 3 contains an in-depth presentation of our motivating application, which is an on-line bookstore with millions of visits and thousands of purchases per day. Subsequently, this chapter contains a discussion concerning related work on balancing QoS and QoD. Using examples from the bookstore application, shortcomings of current related work are identified and explored in depth.

Chapter 4 presents the conceptual framework of our work: (a) the two novel quality metrics of QoL and QoSV, (b) a QoS-centric Control Scheme for regulating QoS and (c) the usage plans of the profiling mechanism for speculating on the user behavior.

Chapter 5 discusses our materialization algorithms. First, it presents the QLS, our QoL sensitive algorithm that balances QoS with QoL by considering link dependencies and the usage plans of uses. Then, it presents the QoSV variation of the QLS algorithm, which provides additional support for set-view dependencies by promoting QoSV. We also identify the problems caused by invalid links in cached fragments and discuss methods of handling them.

Chapter 6 contains the description of the experimental platform that we built in order to evaluate the proposed algorithms. Subsequently, the methodology through which the experiments were conducted and our findings are presented. Finally, the challenges encountered during the design, implementation and testing of our platform are discussed.

Stavros Papastavrou

Chapter 7 summarizes the contributions of the present dissertation and discusses implementation and applicability issues. It also explains how existing applications could benefit from adopting our approach and finally, refers to future work.

Stavros Papastavrou

Chapter 2

Background and Related Work

In Section 2.1, we present various approaches and technologies that focus on enhancing QoS since the mid 90's. The complete review along with a detailed comparison of the approaches can be found in [92]. In Section 2.2, we discuss various propose metrics of measuring QoD in the context of user-driven and data-driven Web database applications.

2.1 Related Work on Enhancing QoS

QoS refers either to the user-perceived or the server-perceived latency. In the first case, the QoS is the elapsed time from the moment the user submits a request for a dynamic page to the time of its delivery at the user side. In the second case, it is the elapsed time from the moment the user request arrives at the Web server to the time the server transmits the dynamic page to the user.

QoS was the early focus of research in Web database applications due to the increased resources that content generating demanded. Work in [11] identifies the performance bottlenecks on the n-tier architecture (Figure 1) for various classes of Web database applications. The findings suggest that the database side is primary responsible for bottlenecks relating to e-commerce

Stavros Papastavrou

and catalog-based applications. For less processing-hungry applications, such as on-line news and multimedia applications, the bottleneck shifts toward the proxy side.

As mentioned in Section 1.3, work on promoting QoS has evolved around three principles: caching, fragmentation and polymorphism. According to the principle of fragmentation, the template file defines separate content areas in a the dynamic Web page called content pieces or fragments, which can be processed separately. QoS is enhanced since only specific fragments can be materialized on every request for a dynamic Web page. According to the principle of caching, materialized content fragments may be cached on the server side, proxy or event at the user side. QoS is enhanced since computational resources are spared when content fragments are reused. Finally, according to the principle of polymorphism, content fragments are assembled together and delivered to the users in various arrangements. In this way, an extra low-cost layer of dynamism is added since the dynamic arrangement of fragments does not require content to re-materialized.

In Figure 10, we express the three principles as the axes of a 3-dimensional mapping structure, which we call the *CFP Framework* [91]. The usefulness of the CFP framework is that, it provides a visual representation of how related research on enhancing QoS has evolved along the three aforementioned principles.

On the CFP Framework, the implementation of a principle is "fine-tuned" or "improved" towards its corresponding edge of the structure. The implementation of caching is improved on the proximity of cached content. The implementation of fragmentation is improved by providing support for isolating and fragmenting any part of a dynamic page. Finally, the implementation of polymorphism is improved by providing support for fragmented arrangement of content. In the center of the framework, we place the CGI as the first ever content middleware, which also provided no optimizations for promoting QoS whatsoever.

Stavros Papastavrou

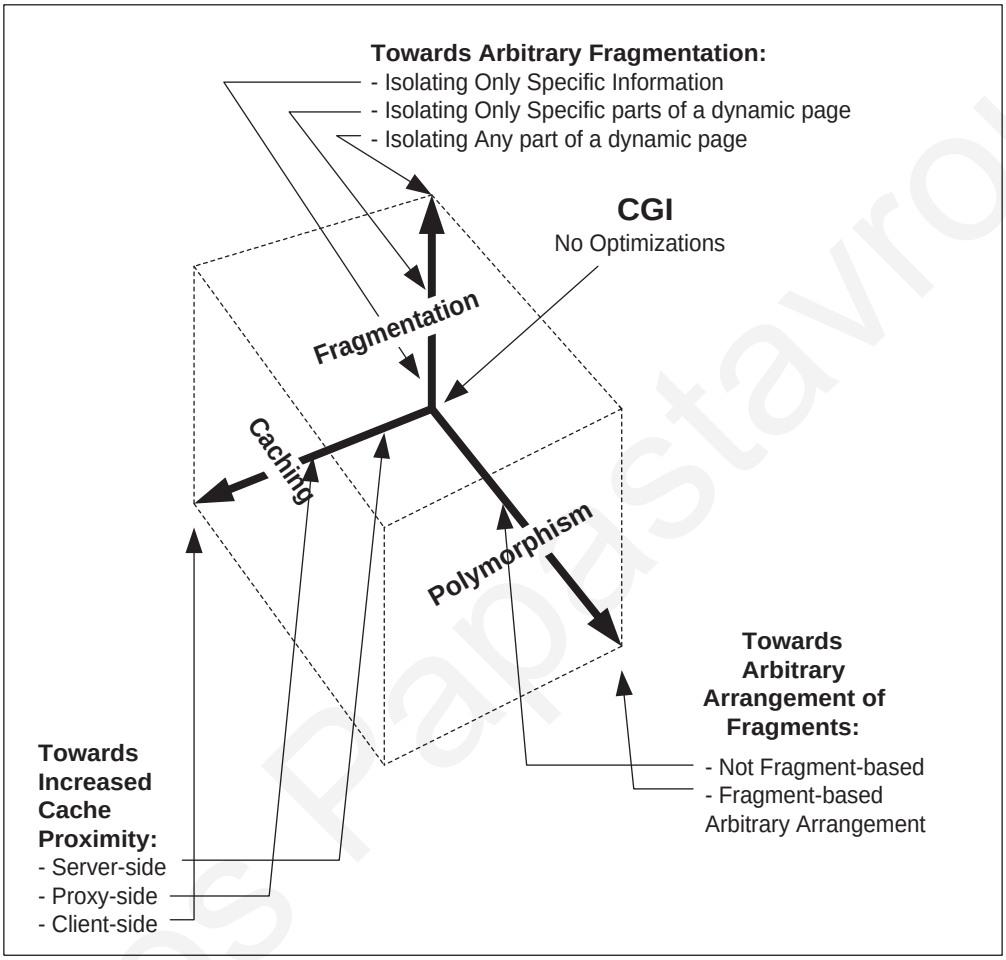

Figure 10: The CFP Framework. The three principles of Caching, Fragmentation and Polymorphism around which QoS research has evolved since the mid 90's. The implementation of a principle is refined (or fine-tuned) towards the edge of the framework. For Caching, implementation refinement is the proximity of cached content. For Fragmentation, implementation refinement is the support for arbitrary fragmentation, which is the ability to isolate any part of a dynamic page. For Polymorphism, implementation refinement is the support for arbitrary arrangements of content fragments.

Stavros Papastavrou

In the rest of this section, we present the related research and technology on enhancing QoS along the three principles. At the end, we map and discuss our findings on the CFP Framework.

2.1.1 Evolution of Dynamic Content Middlewares

The Common Gateway Interface (CGI) was the first ever dynamic content middleware developed for the NCSA Web server [49]. Its major drawback was that, it did support persistent server processes, since CGI processes were designed to terminate right after serving their first HTTP request. Following CGI, FastCGI [3] provided support for server processes to handle consequent HTTP requests *(process persistence)*. Another alternative to FastCGI is mod_perl, which is proven to increase performance over the stateless CGI by an order of magnitude [62]. In addition to persistent processes, mod_perl's performance is due to the employment of persistent database connections to the application server [73]. In conclusion, the pioneer CGI approach and its successors, FastCGI and mod_perl, provided the foundation on which modern content middleware systems are built.

Today, modern scripting languages for content materialization are built on the concepts of the FastCGI and mod_perl. Active Server Pages (ASP) [72] is a technology developed by Microsoft Corp. that support the mixing of Visual Basic code (vbscript) with HTML. ColdFusion [107] is a Java-based product of Macromedia Inc., which uses a tagged-based script code for templates. PHP [56] is a project of Apache Software Foundation that supports a Unix C-like script code. In other less popular approaches, such as Java Servlets [95], direct template file processing is substituted by runnable objects that generate HTML with or without the existence of a template file.

Performance depends heavily on the middleware's implementation. The work in [33] compares various heterogeneous middlewares and reaches to the conclusion that C-based middlewares

Stavros Papastavrou

outperform the Java-based ones. It verifies, however, the programmability of the Java-based middlewares for generating content. In [88], a detailed study of all the Java-based middlewares for dynamic content generation is found, along with a thorough comparison in terms of performance and programmability.

2.1.2 Other Related Architectures

Beyond the well-accepted and practical multi-threaded architecture of Web servers, there have been significant efforts for the development of more efficient architectures to boost content generation. Flash, presented in [86], is a portable event-driven Web server that has been demonstrated to outperform both the single-process event-driven Zeus Web server[5] and the multi-threaded/multi-process Apache Web server [1]. Its portability lies on the fact that it uses standard APIs, found in any modern operating system. Flash, however, was originally designed to accelerate the delivery of static content and, yet, there has been no adaptation of it for delivering dynamic content.

More recently, the authors in [111] propose the use of an event-driven Web server and introduce the notion of a *stage*. According to [111], a Web application (i.e., a Web server) is built as a network of explicit computation stages connected by explicit queues aiming at supporting massive concurrency and simplifying the construction of Web services. Similarly, the authors in [68] employ staged computation in Web servers, which replaces threads and introduce a more sophisticated task scheduling mechanism. To the best of our knowledge, there is no system that uses those architectures for dynamic content generation.

2.1.3 Fragmented Content Materialization at the Server Side

The principle of fragmentation is introduced with the Server-Side Includes (SSI) [4]. According to SSI, specific parts of a Web page are isolated and materialized every time the Web page is

Stavros Papastavrou

requested. Examples include counters, the time at the server, information about the requested file, as well as custom built information.

Work in [34] proposes a more general form of fragmentation that supports the dissection of a dynamic page into distinct fragments that are assembled according to a template file (example in Figure 11). A fresh version of a fragment is materialized every time its underlying data objects are modified, with the use of database triggers. With the fresh fragments in place, a dynamic page can be either immediately delivered or cached. This approach supports arbitrary fragmentation, however, it provides caching only at the granularity of a page. It is more suitable for less interactive Web applications such as portals and news sites, since the generation of content is data-driven.

More recently, [90] propose a technique for accelerating template parsing and execution, by materializing the dynamic fragments of the template in a concurrent fashion. It is worth mentioning that the identification of the fragments takes place at run time (during parsing) and requires no a-priori compilation or special handling of the template. This approach supports arbitrary fragmentation and immediate execution of fragments with no caching. It is more suitable for interactive Web applications such as e-commerce, where content generation is user-driven.

Currently, modern scripting languages, such as ASP and PHP provide full programming level support for fragmentation, as developers are provided with language specific tools and tags to isolate any part of a template.

2.1.4 Content Caching at the Server Side

Server-side content caching boosts the generation of content by eliminating redundant server and database workload. There are many interesting approaches for server-side caching that vary mostly on the granularity and level of caching. In [55] and [52], the caching of dynamic documents

Stavros Papastavrou


```
<HTML><HEAD>...</HEAD>
<BODY>
  <TABLE>
  <TR><TD> \\ script code block for fragment #1 </TD></TR>

  <TR><TD> \\ script code block for fragment #2 </TD></TR>

  <TR><TD> \\ script code block for fragment #3 </TD></TR>
  </TABLE>
  </BODY>
</HTML>
```

Figure 11: Implementing a Fragmentation in the Dynamic Search Page of Google. In this simple example, static HTML defines the layout of the script code blocks corresponding to the three fragments.

Stavros Papastavrou

at the granularity of a page is proposed for early content middlewares such as CGI, FastCGI, ISAPI and NSAPI.

Extending their work in [52], the authors in [104] propose a caching Protocol that can be implemented as an extension to HTTP. The proposed protocol allows for content middlewares, such CGI and Java Servlets, to specify full or partial equivalence among different URIs (HTTP GET requests). The equivalence information is inserted by the content middleware into the HTTP response header of a dynamically generated page and stored at the caching module along with the cached page. For example, the URI

"http://www.server.com/LADriveTo.php?DestCity=newyork"

instructs the content middleware to generate a page with driving directions from Los Angeles to New York. Prior to transmitting the result page, the middleware inserts the

"cache-control:equivalent_result=Dest=queens"

attribute in the HTTP response header. The caching module will cache the page, transmit it to the user and store the cache-control directive for future use. A subsequent user request for the same URL, but for a different DestinationCity value, will be evaluated by the cache module for a possible match with the value of "queens" or "newyork". If a match is found, then the cached page is transmitted to the user.

2.1.5 Caching Content at Finer Granularities

To achieve greater reuse of cached content, caching at finer granularities is proposed. The authors in [114] suggest the caching of static HTML fragments, XML fragments and database query

Stavros Papastavrou

results. This approach, however, applies to Web applications that follow a strict declarative definition and follow a certain implementation only. In addition, caching cannot be applied to random parts of the dynamic page and, in extend, consequently, does not allow for arbitrary fragmentation.

A more general and flexible approach for fragment caching is introduced later on in [41] and studied more thoroughly in [40]. According to this method, caching can be applied to an arbitrary fragment of a template by first wrapping it around with the appropriate tags (explicit tagging). Similar, XCache [7] is a commercial product that installs as a plug-in on popular dynamic content middlewares and supports fragment caching of any type. Also, the Cold Fusion content middleware provides tags for explicitly defining the page fragment to be cached. For example, the following coding:

```
<cf_cache refresh-rate=60>...script and HTML...</cf_cache>
```

caches an arbitrary fragment that refreshes every 60 seconds. In addition, modern scripting languages, such as ASP and PHP provide programming level support for fragment-level caching at the server.

The performance gains of fine-grained content caching at the server side on high server workload, are also shown in the "WebView Materialization" approach found in [61, 63]. This study explorers in depth the performance of various content materialization methodologies in the present of continues data updates on the application database (i.e., updates on stock-related database). We elaborate on these studies in the next chapter, when we present related work on balancing QoS with QoD.

Stavros Papastavrou

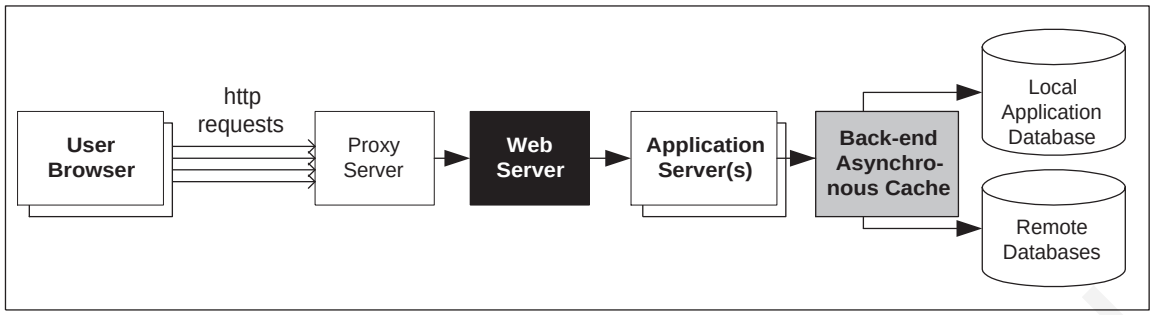

Figure 12: Middle-tier Fine-grained Caching using an Asynchronous Cache

2.1.6 Middle-tier Fine-grained Caching

Another serve-side approach is the caching of materialized content fragments between the application server and the database server [66, 65]. According to this approach, widely known as *middle-tier caching*, an asynchronous cache module exists between the application server and the database server (Figure 12). The purpose of this extra module is to intercept requests for dynamic content fragments and either serve them immediately from its cache, or materialize them from scratch by querying the application database.

The main difference between this approach and server-side caching is that, the asynchronous cache module periodically selects a specific set of content fragments for materialization and storing at the asynchronous cache, so that they are ready for immediate reuse the application server. This set of fragment is selected according to QoS-QoD balancing metrics that we discuss in the next chapter. Middle-tier caching is also endorsed by IBM and Microsoft [25, 67].

2.1.7 Proxy Caching

Proxy caching is the most popular approach for faster delivery of reusable static content such as static HTML pages and media files [108]. A proxy degrades bandwidth consumption by eliminating unnecessary traffic between users and servers, given that it is strategically located. It has

Stavros Papastavrou

been identified that the usual hit ratio for proxy caches is around 40% [113]. Another study, however, shows that even when proxies are employed, approximately 40% of the original volume of Web traffic is still unnecessarily generated [105].

Early research conducted in [104], proposes the caching of dynamic content at the granularity of a page by using the Dynamic Content Caching Protocol as previously discussed. The caching protocol is applicable for both server-side and proxy-based caching and works by allowing the manipulating of HTTP header information and URL query string parameters (GET variables).

Another interesting approach for caching whole dynamic pages is found in [74]. Similar to the caching protocol approach discussed earlier, this one suggests that the proxy server be allowed to examine the HTTP POST variables that are submitted as part of a user HTTP request for a URI. In brief, the proxy server attempts to reuse cached SQL query results by looking up on a predefined mapping. This mapping relates (a) the HTML form fields that are submitted with a URI request with the SQL query that uses those form fields as input. Two strong points of this work is that (a) the proxy can extract and reuse portions of cached query results, if necessary, to satisfy future requests and (b) it can add-on to a cached query result on demand by negotiating with the Web server. Since the HTTP post variables are generated from HTML form fields, this approach is called *form-based*.

2.1.8 Fine-Grained Proxy Caching

Extending proxy caching, caching at the granularity of a fragment is proposed for proxy caches. According to fine-grained proxy caching (Figure 13), a template file is cached at the proxy server whereas its dynamic fragments are either reused from the proxy or fetched fresh from the Web server.

Stavros Papastavrou

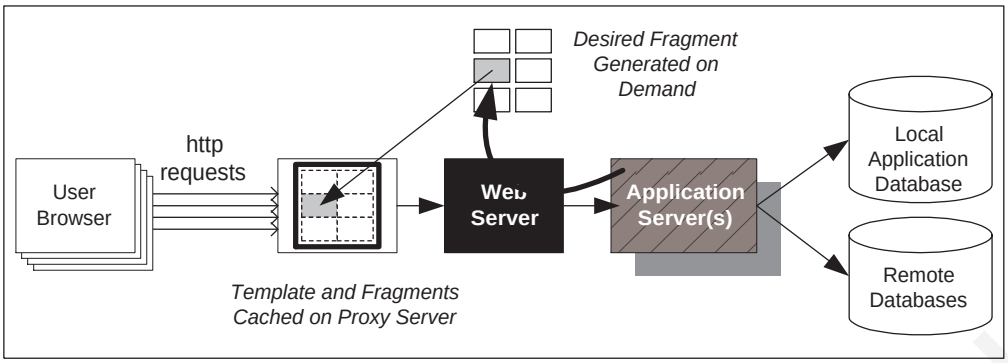

Figure 13: Fine-Grained Proxy Caching. The template of the dynamic Web page is cached on the proxy server. The missing fragment is generated on and fetched on demand from the server-side. Once fetched, it substitutes the corresponding invalid fragment and the complete page is sent to the user.

To facilitate fine-grained proxy caching, Edge Side Includes (ESI) was introduced as a standard for caching page templates along with their fragments on proxy servers [2]. According to ESI, the dynamic fragments of a page are explicitly marked using tag-based macro-commands inside a template file. An ESI-compliant proxy server must provide support for parsing the cached template file and executing macros that decide whether a fragment should also be retrieved from cache, or pulled from the original server. ESI macros have access to a user's HTTP request attributes (cookies, URL string, browser used) in order to choose between fragment alternatives. An example of that would be the identification of the user's browser version in order to pick the appropriate fragment that meets the browser capabilities.

ESI is a key component for Content Distribution Networks (CDN), a popular caching approach that supports the leasing of cache space on a service-based network of interconnected proxy servers. A typical CDN employs a set of proxy servers strategically arranged by geographical, or network location. It is noteworthy that for a Web site to be registered and served by a CDN network, an off-line procedure of updating the templates of the Web site is required. A thorough survey on the practices and performance of CDNs can be found in [57].

Stavros Papastavrou

ESI extends the approaches that support arbitrary server-side caching ([41], [40] and scripting languages such as PHP, ASP and ColdFusion) by moving fragment caching from servers to proxies. ESI also provides some basic support for dynamic fragment arrangements through the use of a tag-based scripting language, a practice that we identified as polymorphism in [91].

A more recent study in [99] proposes a different approach to content fragmentation and its caching. Instead of using explicit fragmentation techniques such as tagging (ESI, ColdFusion), it proposes an automatic fragment detection framework which isolates the "most beneficial" content in terms of caching. More specifically, the fragmentation is based on the nature and the pattern of the changes occurring in dynamic Web pages and its potential use by more users. This approach was found to improve the efficiency of disk-space utilization at the proxies and reduce the load on both the network and the origin server.

2.1.9 Fine-Grained Caching at the User-Side

Surprisingly, the notion of assembling a dynamic page away from the original content middleware was firstly introduced in [45] not for proxy caches, but for user browsers. The proposed technique, called HPP (HTML Pre-Processing), requires from the user browsers the extra functionality of caching and processing a template file. The file contains blocks of macro-commands that are processed prior to rendering the dynamic page. Each macro-command block generates from scratch a page fragment by manipulating local data. This idea can be viewed as the client-side equivalent to Server-Side Includes discussed earlier.

Extending their early work in [45], the authors in [98] propose the Client-Side Includes (CSI) by merging HPP and ESI. In order to provide support for CSI in Web browsers, the authors propose a generic downloadable wrapper (plug-in) that uses JavaScript and ActiveX. The wrapper pulls and caches at the user side the template and fragments that are associated with a requested dynamic

Stavros Papastavrou

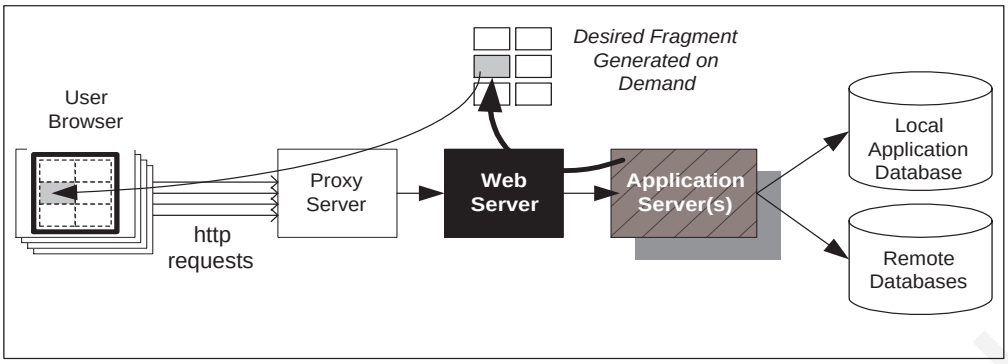

Figure 14: Fine-Grained User-side Caching. The template of the dynamic Web page is cached on one every user. The missing fragment is generated on and fetched on demand from the server-side. Once fetched, it substitutes the corresponding invalid fragment on the user's browser.

page, assembles them together according to the ESI directives in the template and finally renders the page. According to the authors, CSI is suitable for "addressing the last mile" and it better suits low-bandwidth dial-up users.

The original CSI approach, as proposed in [45], employs full caching and targets low-bandwidth users. However, the approach does not provide full fragmentation, since the cached parts of a document cannot be reused by other ones. The improved version, as proposed in [98], supports full fragmentation by allowing arbitrary content fragments to be cached and reused by any templates at the user browser. Figure 14 illustrates the practice of user-side fine-grained caching as an extension of the proxy-side equivalent.

2.1.10 Polymorphism: A second Dimension of Content Dynamism

Caching at the fragment level requires the existence of a page layout or template that defines a strict arrangement for cached fragments. The principle of polymorphism suggests the arbitrary arrangement of fragments inside a template file.

Stavros Papastavrou

The approach found in [42] provides full support for proxy-side arbitrary polymorphism by switching between a list of available templates the relate to specific dynamic page (Figure 15). According to this approach, a user request for a dynamic page (e.g., www.server.com/ page1.php?id=2) is always routed to the original Web server and causes the execution of the original script (in this case the page1.php). This execution is necessary for determining the desired template for page1.php at run time. The selected template is then pushed to the proxy server (if not cached there) and parsed for identifying which fragments should be reused from cache and which should be requested fresh from the server. The performance analysis conducted in [42] demonstrated solid bandwidth reductions when applying fragment caching. However, performance analysis for other critical metrics, such as scalability and responsiveness, remains to be seen. We believe that both the necessary routing of every request to the origin content server and the invocation of the original script can hurt user response time and server scalability, respectively.

Limited support for polymorphism is found in ESI. Instead of choosing from a template pool, basic ESI branching commands can reorganize parts of the layout inside a template, according to user preferences (Figure 17). [42] extends ESI by providing arbitrary support for polymorphism at the proxy. This support, however, is achieved through vendor-specific code that requires extra proxy functionality.

Arbitrary fragmented polymorphism is also realized at the user side with the use of Javascript and involve user interaction. Javascript functionality can be built behind content fragments so that users can drag the fragments into various locations in the containing dynamic page. On example is the BBC news Website.

As stated earlier, polymorphism can be realized as a stand-alone practice without the use of independent content fragments with dynamic arrangement. This practice is called *flat polymorphism*. In this case, the script that generates the entire dynamic page must provide the functionality

Stavros Papastavrou

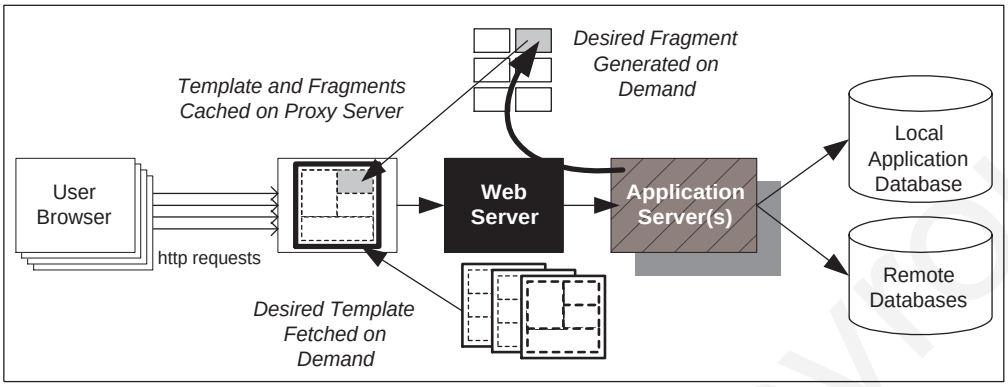

Figure 15: Polymorphism With the Use of More Than one Templates. Missing fragments and desired template are both fetched from the server side.

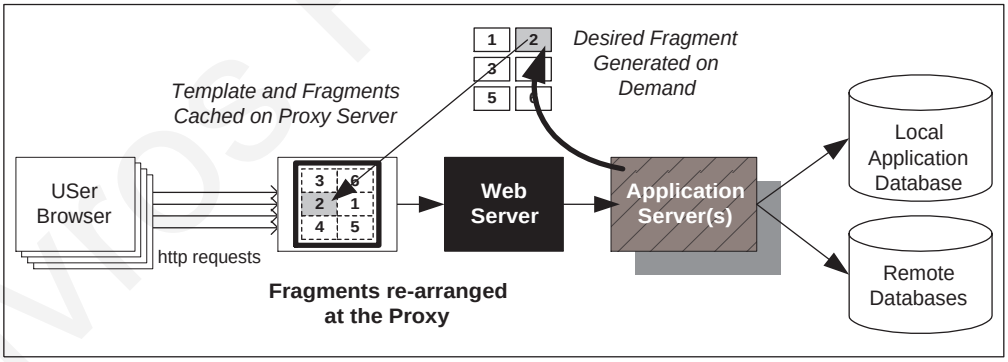

Figure 16: Polymorphism by Re-arrangement of Fragments. Missing fragments and desired template are both fetched from the server side. Fragments are dynamically re-arranged using proxy support.

Stavros Papastavrou

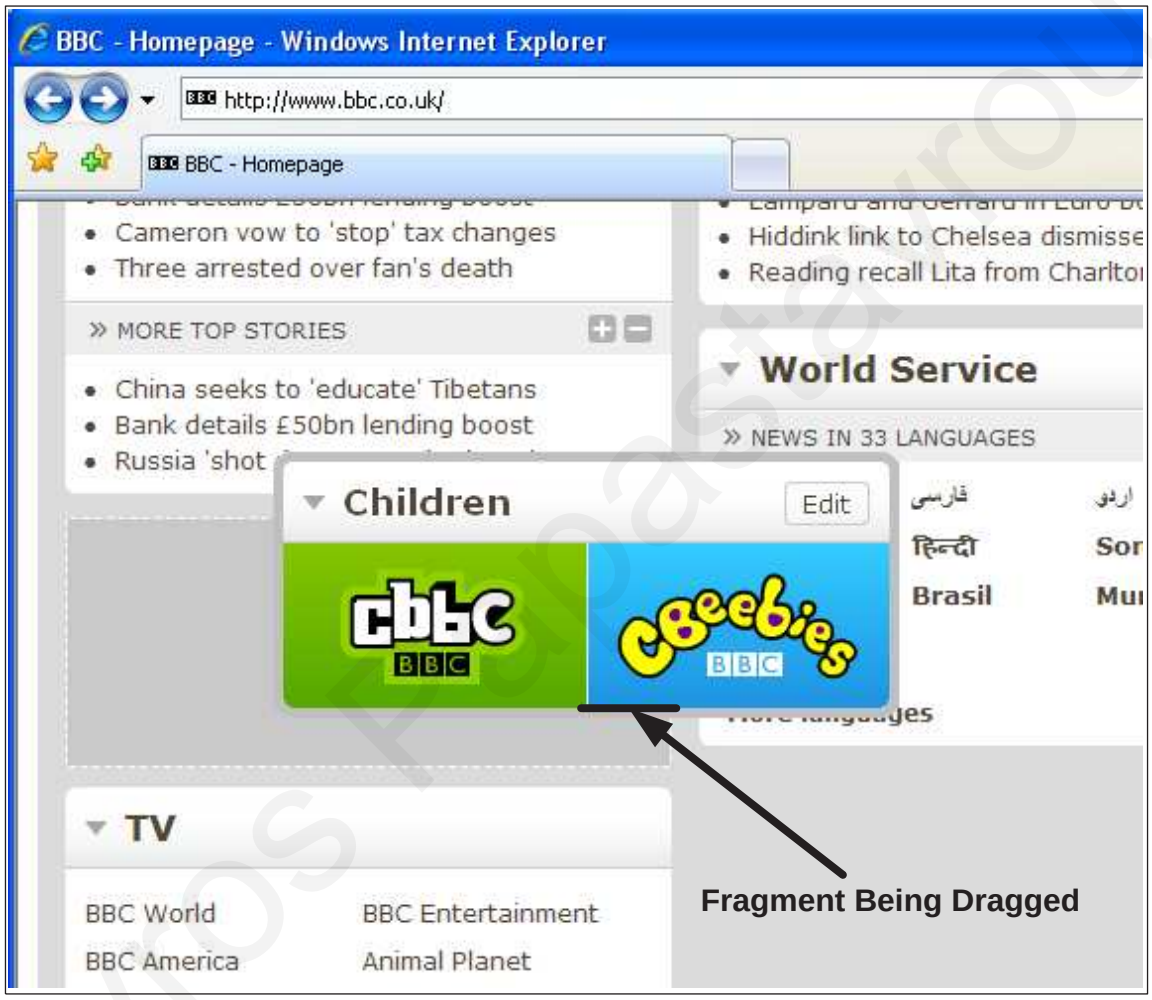

Figure 17: Polymorphism on the User Side with Support from Javascript. Web users re-arrange the content fragments by drag-and-drop.

Stavros Papastavrou

that internally rearranges the layout of the content according to user preferences. Unlike the approach discussed in [42], flat polymorphism is not suitable for content caching, since the entire script must execute on every user request and content fragments cannot be isolated and cached separately. Support for this form of polymorphism is provided by any scripting language and is entirely left on the programmer to implement.

2.1.11 Caching with Delta Encoding

Delta encoding is a popular technique for efficiently compressing a file relatively to another one called the "base" file [54]. This is achieved by computing and storing the difference between the file being compressed and the base file. Streaming media compression, displaying differences between files (the UNIX diff command) and backing-up data are common applications of delta encoding.

Under the assumption that consecutive user requests for a specific URI would generate a sequence of moderately different dynamic pages, Delta encoding can be exploited as an alternative for caching dynamic content. [96] proposes the caching of a base file for each group (also called Class) of correlated documents i.e., pages that share a common layout. With the base file cached, the next user request would force the content middleware to compute the Delta between the new dynamic page (that the user would normally receive) and the base file. The computed Delta is then transmitted from the content middleware to the side where the base file is cached for generating the new dynamic page. Subsequently, the result is transmitted to the user. An interesting feature of this so-called "class-based delta-encoding" approach, is that the base file can be cached either at the server-side, proxy-side, or even at the user browser itself as long as the required infrastructure exists. In the latter case, Delta encoding benefits could low-bandwidth users. [96] demonstrated

Stavros Papastavrou

solid bandwidth savings and reduced user perceived latency. However, the performance gains reduce the average system throughput to 75% due to increase CPU overhead of computing the deltas. Nevertheless, Delta encoding, in association with fine-grained caching, for caching dynamic Web content is an exciting open topic of research.

2.1.12 Active Caching

The notion of Active Caching refers to the ability of a caching middleware to manipulate cached content instead of requesting fresh versions of it from the server. The approach found in [31] piggybacks a Java object into a dynamically generated document, which is then cached at the proxy. The proxy provides a Java runtime environment in which that object executes, in order to modify the dynamic parts of the cached document according to a user's preferences. Examples of document modifications include advertising banner rotation, logging user requests, Server Side Includes execution and even Delta compression. In addition, the Java object can personalize cached documents by retrieving personal information from the application database at the server side. Data chunks of personal information are kept by the object for future reuse.

Building up on this approach, a more general form of dynamic content caching is suggested in [75]. This one is very similar to the "form-based" approach discussed earlier, in the sense that the Java object manipulates the HTTP post variables (the Form input) for generating the dynamic parts of the cached document.

Active caching approaches combine the advantages of proxy-side caching while providing some support for fragmentation. They do not employ full fragmentation since the fragments are not decoupled from the template (are not stored separately) and therefore cannot be cached and reused.

Stavros Papastavrou

2.1.13 Multicasting of Dynamic Web Content

Early studies have found that the popularity distribution for documents in a Web site follows a zipf-like distribution [13, 28] with parameter α less than 1. Most recent studies place the parameter α between 1.4 and 1.6 [85, 58], which implies that less documents (about 2%) account for the most accesses (approximately 90%).

The above findings, in conjunction to advances in wide area networks (WAN), have driven researchers to suggest the multicasting of the most popular documents in a Web database application. Multicasting of content, introduced in the late 90's, refers to the delivery of content to a group registered Web users, using the most efficient strategy over a network channel, utilizing each link of the network only once and exploiting the locality of document requests [16].

Subsequently, the work in [44] introduces the HTTPM protocol for the transfer of "hot" dynamic documents over multicasting channels, in conjunction with the use of HTTP for unicasting (targeting only a specific user at a time) less popular pages. Other studies have proven the advantage of multicasting popular dynamic documents in terms of bandwidth [15] and QoS [18, 70, 17, 19, 116].

2.1.14 Modern Approaches: Active XML and AJAX

An approach similar to active caching is Active XML (AXML) [8]. An AXML template file, usually cached at a proxy, includes calls/references to Web Services of the form

```
<sc>rentdvd.com/getPoPularDvdList()</sc>
```

that return the dynamic parts of the template. To support AXML, a runtime is required at the location where the templates are cached. Its responsibility is to parse the templates and trigger the calls to the referenced Web services.

Stavros Papastavrou

AXML provides an alternative form of fragmentation in which the template of the dynamic page is cached at the proxy and the fragments themselves are substituted by function calls. A strong point of AXML is that the references to XML services embedded in a template file can be reused by other templates.

Finally, the most recent approach for dynamic content generation is Asynchronous Javascript and XML, better known as AJAX [93] and is classified into fine-grained user-caching approaches. Similar to SSI, ESI and CSI, the AJAX approach suggests the inclusion of dynamic content areas in a Web page, which is designed to stay loaded for a certain period on the user's browser. The dynamic parts of the Web page are regenerated by first loading XML data from the server side and then by rendering it on user browser using Javascript. Unlike its predecessors, AJAX does not require extra functionality at the user, as this is provided by any modern Web browser that supports Javascript and cascading stylesheets. AJAX techniques have been used before the widespread of XML, however, the recent standardization was a necessity due to the great variety of methods.

Web database applications based on AJAX differ from traditional Web-based approaches since the underlying idea is that an application be run inside a single Web page. The most typical example is that of Google's Gmail [112], where a basic template remains loaded on the browser and certain areas change dynamically. The big advantage of AJAX applications is increased performance, since only the necessary data is loaded on every user request. The major drawback, however, is that AJAX applications are stateless, in the sense that the notion of "back" and "forward" for the browser does not exist. In other words, a user can loose the whole setup and state of the application if the "forward" or "back" buttons are pressed.

Stavros Papastavrou

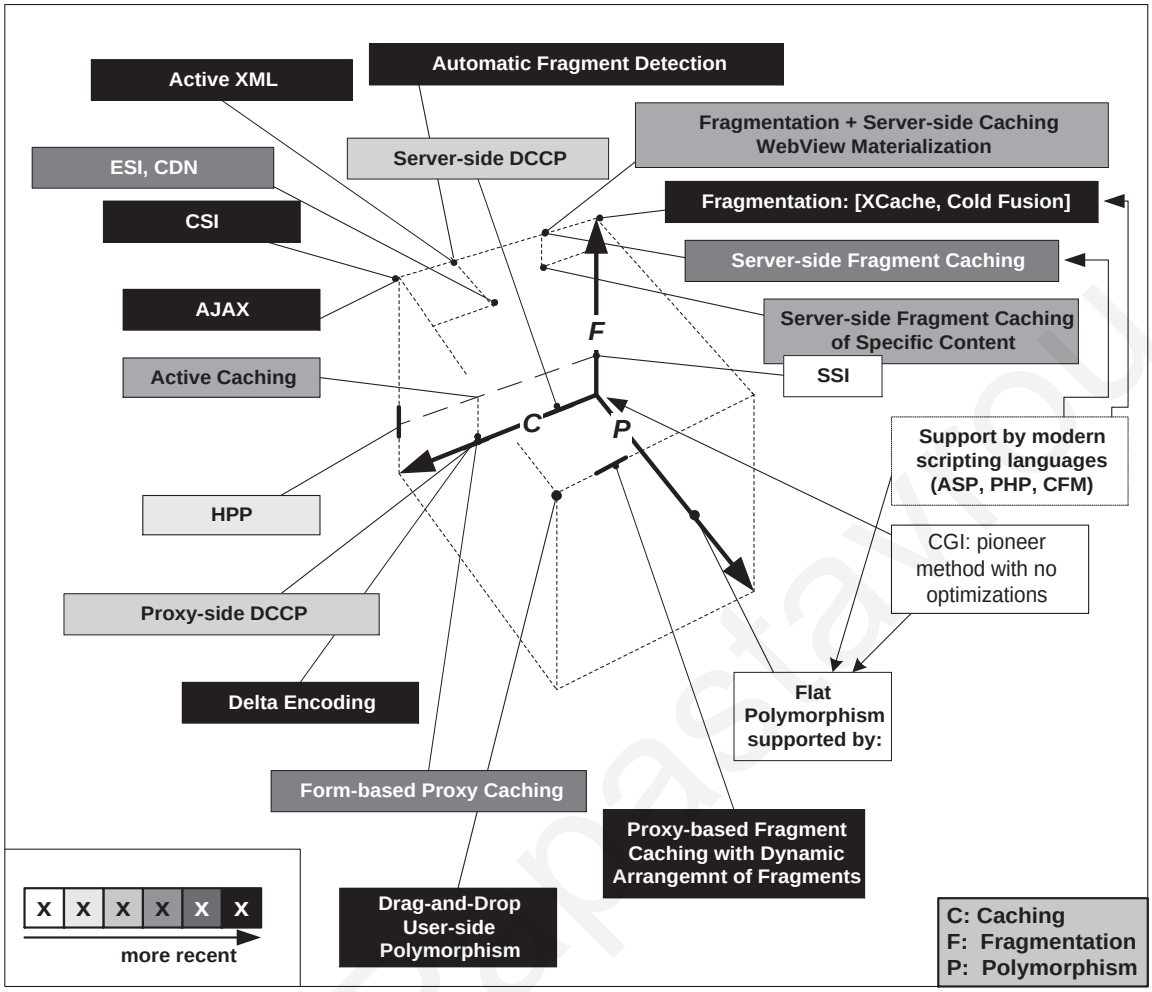

Figure 18: Related Research and Technology on Promoting QoS.

2.1.15 Discussion

Related research and technological approaches on enhancing QoS, which employ at least one of the three principles of caching, fragmentation or polymorphism, are plotted on the CFP Framework (Figure 18). The approaches are shaded using a dark-grey background, according to their chronological appearance. At the center of the framework, we place the CGI as the primary middleware that employed no optimizations on QoS.

As evident in Figure 18, more recent approaches are located toward the edge of the framework. Therefore, the evolution of research on promoting QoS has been toward refining the employment

Stavros Papastavrou

of the three principles, as well as finding ways to combine them. In other words, QoS is promoted by caching content closer to the users (for faster response times), at finer granularities (for increased caching reuse) and served to the users under various possible arrangements (to support personalization).

2.2 Related Work on QoD Metrics

Quality of Data (QoD) has been studied thoroughly in the context of Web database systems. Subsequently, QoD has been given many interpretations with respect to the nature and degree of deviation of the materialized content from the database objects due to data updates. The proposed metrics, some of which are discussed below, suit to different application semantics and needs. Below, we first discuss popular QoD metrics in the context of database-driven and user-driven Web database applications. Then, we present briefly other assorted metrics.

2.2.1 Metrics for Database-driven Applications

In the context of Web database applications, which exhibit frequent updates on the application database, a fragment of materialized content is considered fresh as long as the database objects that were queried or manipulated for the purpose of generating the content fragment are still in the same state. A real-time stock information application, as studied in [61, 63, 64, 65, 66], is a typical example of such database-driven applications. Once updates occur on the dependent database objects, the corresponding materialized content fragment is no longer considered fresh. In this case, various metrics that capture the freshness (or staleness) of the materialized content fragment are proposed:

In the early 90's, the metric of *Currency* was introduced in [103] in the context of data warehouse systems. It measures the elapsed time from the moment of content materialization, to the

Stavros Papastavrou

time of its delivery to the users. Introduced in the mid 90's, the metric of *Timelessness* measured the time elapsed from the creation or update of the data objects involved in materializing a content fragment, until the moment of its delivery to the users [110].

According to [36], cached content (i.e., a fragment) is as old as the time elapsed since its underlying database objects were modified. The same work introduces the notion of a *fresh* database, according to which a database is fresher than a second one when it has more up-to-date elements. In addition, it introduces the notion of *age* of a database, which measures the average age of data items. The age of up-to-date (fresh) items is zero, as opposed to outdated items, which have an age analogous to their modification time. This approach was designed to facilitate search engine crawling and data warehouses.

The work in [53] considers as age the number of updates on a database object. Read-only queries on a data object are tagged with a tolerance metric that relates to the maximum number of allowed updates on the object. In addition, an object must be pushed to the users when a specific threshold on the number of updates is reached. This approach was designed to facilitate applications such as inventory systems and airline reservations, whose users are roaming in low-bandwidth networks. Similarly, the work in [47] uses a three-way cost model that measures the insertions, deletions and modifications on data items. This model is studied in the context of optimizing query processing for updating materialized views.

In the context of broadcast-based data dissemination, the work in [94] introduces the notion of *currency interval* as the metric for data freshness. According to this work, the currency interval of a data object which is broadcasted is the period that the object is considered fresh or "valid". If no update has occurred on the object, then its currency interval is "infinite".

Stavros Papastavrou

2.2.2 Metrics for User-driven Materialization

In the context of user-driven Web database applications, which materialize content as a result of user-submitted personalized requests, *a content fragment is considered fresh as long as it is always materialized*. In the present dissertation, this is the assumed QoD metric. On-line bookstores and product customization sites are applications that fall into this category. In this case, content fragments relate to the *Virtual WebViews*, as introduced in [61, 63], according to which a specific category of content fragments must be materialized on every user request to meet the application semantics.

In this context, a dynamic Web page is considered fully fresh as long as all its content fragments are materialized on every user submission. The use of cached fragments compromises/reduces the overall freshness of a dynamic page, according to the weighted importance of the cached fragments.

2.2.3 Assorted QoD Metrics

According to [10], other metrics on QoD are:

- *Data Uniqueness*. It defines the degree to which two or more data items have values that do not conflict each other [77].
- *Data Consistency*. It expresses the degree to which data satisfies a set of integrity constrains [100].
- *Data Accuracy*. It measures the precision of an application database to the real-world values. Accuracy can be either semantic [110], syntactic [81] or precision-based [100].
- *Data Completeness*. It measures the degree to which all information relevant to an application domain is recorded in a database [48, 24, 80]. Two types of data completeness are

Stavros Papastavrou

identified: Coverage Completeness and Density Completeness. The former considers the existence of the required data entities and the latter considers the presence of the required data values. A data value is considered present with the corresponding record field is not null.

2.3 Chapter Summary

Related research and technological advances on enhancing quality of service (QoS) for Web database applications were presented, revealing that the evolution of research is closely related to the three principles of caching, fragmentation and polymorphism. The trend was the refinement and combination of the implementation of the three principles, to meet the characteristics of modern Web applications. In other words, dynamic content tends to be cached closer to users, at finer granularities called fragments and served to users under various possible arrangements.

Popular metrics for measuring QoD of materialized content were also presented. For database-driven Web applications, the metrics relate to changes on the application database objects. For user-driven applications, content needs to materialize on every user request to be considered fresh. This is what is considered as the QoD in this dissertation.

Stavros Papastavrou

Chapter 3

Current Quality Metrics for Web databases

In this chapter, we elaborate on the shortcomings of the current approaches to balance QoS and QoD and present the intuition behind our semantic-based approach. We do so by first presenting our assumed system model and with the help of a motivating application.

3.1 System Model

In this section, we present the system model of the present dissertation which supports web database applications that are user-driven, serve personalized content and do not have frequent updates on the application database.

Our system model is based on the typical architecture shown in Figure 1 consists of the user browsers, the proxy server, the web server, the application server and the application database. All the component modules in the architecture may have a cache, however, in this present dissertation we focus on the cache of the application server, which is the module responsible for content materialization and caching, as well as regulating QoS by reusing cached content. Moreover, we do not assume a common shared cache across all user sessions but rather we assume that each user session is allocated its own cache for its fragments. We distinguish between individual user sessions

Stavros Papastavrou

with the use of web cookies in user browsers [59], a mechanism for individual session tracking and state maintenance that has enabled web applications to include session-specific content such as shopping carts.

In the architecture, the web server is the entry point of the web application. Requests for static content, such as images and cascading stylesheets, are served immediately from the web server. Requests for dynamic web pages are routed to an application server that executes a corresponding template file. This file includes script blocks that relate to content fragments. The application server can either materialize a fragment from scratch, or reuse from cache the content of a previous materialization. The materialization of a fragment includes queries to the application database and formatting of their results with HTML. Finally, the content for all the fragments, cached or freshly materialized, is put into place according to their template file and transmitted to the user through the web server.

According to our system model, content fragments that are reused from cache do not have any staleness requirements. In other words, a fragment that is materialized on every user request for its containing dynamic page, is considered fresh. On the other hand, there is no guarantee of freshness for a fragment that is reused from cache and hence does not contribute to the overall freshness of its containing dynamic page.

Definition 1. (Freshness of a Content Fragment) *A dynamic web content fragment, which is transmitted to the user upon request on its containing template, is considered fresh if it has been materialized according to the user's submitted parameters.*

Definition 2. (Freshness of a Dynamic Web Page) *A dynamic web page is considered fresh if all the fragments of its corresponding template are fresh.*

Stavros Papastavrou

Although, in general, cached fragments are considered stale, under certain conditions that relate to set-view dependencies, as we explore in the next chapter, cached fragments may contribute to the set-wise consistency of a dynamic page and without negatively impacting its freshness.

Fragment materialization in our model is analogous to virtual WebViews introduced in [61] and explored thoroughly in [63], in the context of web database applications with frequent database updates. Virtual WebViews are analogous to content fragments that must be materialized on every user request for their containing template. However, the main difference between the fragments in our system model and those in WebViews is their contents and usage. In the present dissertation, the fragments are assumed to contain HTML FORM and URL links with dynamic parameters [20] that enable a user to navigate between dynamic web pages. We elaborate on dynamic URL and HTML FORM parameters by explaining their two implementations:

URL Links First and most popular are the parameterized URL links. Those links point statically to the target page's template file, i.e., ...com/viewBook.dyn and append dynamic URL parameters of the form

"?bookid=2345&action=changeQuantity&value=-1".

Those parameters are set on every fragment materialization.

FORM Links The second type of linking between pages is through HTML Forms, in which users type in and submit search keywords. The target template is defined in the action parameter of the FORM.

Consequently, we assume that a fragment reused from cache always contains invalid/outdated links. This is because the parameters of links in a fragment are not hardwired since they are

Stavros Papastavrou

Dynamic Web Page	Template	Abbreviation	Popularity
Search Page	search.dyn	S	50%
View Book Page	viewBook.dyn	V	25%
Shopping Box Page	shopBox.dyn	B	12.5%
Used Books Page	used.dyn	U	6.125%

Table 2: The four most Popular Dynamic Web Pages of the bookstore Application

dynamically set on every fragment materialization. Therefore, the parameters of links in a cached fragment would refer to a previous user application-specific state and, therefore, would be invalid.

Another important difference between the fragments in our model and WebViews is that there is no caching reuse across users sessions due to content personalization. This has a major impact on the cache storage requirements and, consequently, on the cache replacement strategy since every user session requires its own personal cache storage. As a result, when there is no space in the cache of an individual session, the replacement strategy evicts from cache the fragments that are most recently materialized.

3.2 The On-line Bookstore

Our motivating application is an on-line store with emphasis on books, millions of visits and thousands of purchases per day. We have selected a bookstore application because of its complexity, its typical functionality as a user-driven web database application and, finally, for its popularity. According to Nielsen On-line [6], on average, 41% of Internet users have purchased books. It is also estimated that book purchases have increased by 40% in the last two years. Only in the United States of America, more than 57 million Internet users have purchased at least one book.

Our bookstore store consists of more than 20 different templates of which the four most popular are a search page for finding a book, a view book page for viewing book-specific information, a shop box page for shopping cart handling and a used books page. These four pages account for

Stavros Papastavrou

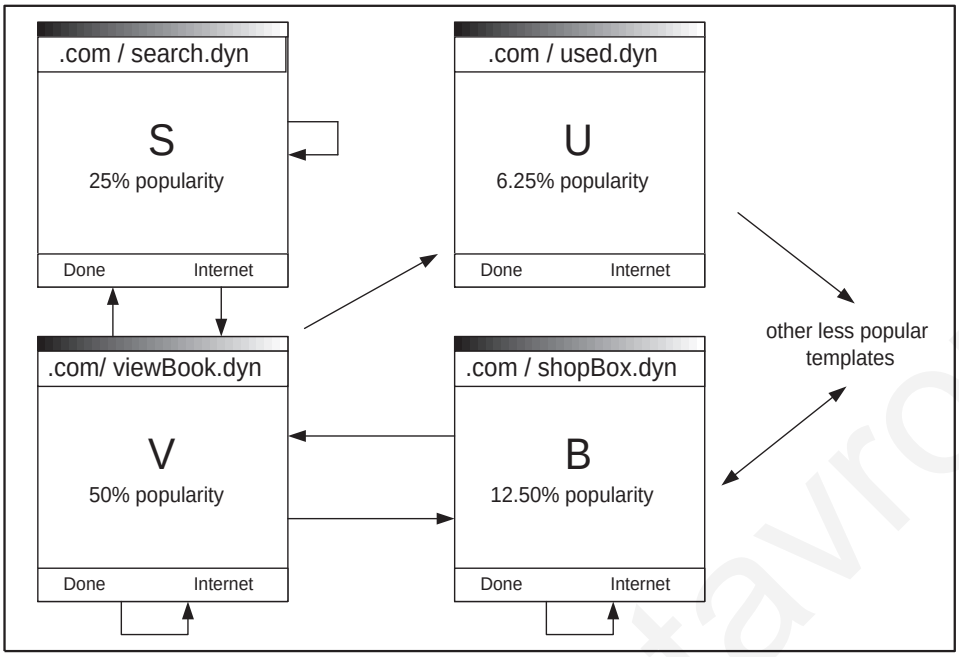

Figure 19: The four most Popular Pages of the bookstore Application with their Navigation Possibilities

more than 95% of user accesses and are summarized in Table 2. Other less popular web pages include editorial reviews pages, feedback pages, user profile pages, as well as check-out specific pages.

Figure 19 illustrates an overview of the bookstore and the possible navigation among its pages. The arrows between templates denote that a user can navigate from one template to another. Figure 20 shows the fragments of the three most popular templates. The arrows denote that a user can use a link or an HTML Form from within the fragment, to navigate to the target template. Some fragments have link dependencies to more than one templates.

Template search.html (S) The users start their book quest by submitting searches through a search page implemented by the `search.dyn` template. The search is done mainly through an HTML form on top of that template and the results are displayed again in the search.dyn page. Users can repeat their search indefinitely. The contents of the search page are search results and suggested book listings. A main fragment $F1_{topresults}$ displays the top results and a secondary

Stavros Papastavrou

below F2$_{allresults}$ displays all the results in a page-numbered way. Two fragments on the left, F3$_{suggested1}$ and F4$_{suggested2}$, show suggested related sponsored results. F5$_{searchform}$ contains the HTML search Form plus a few suggested terms for repeating search. F6$_{resubmit1}$ and F7$_{resubmit2}$ suggest search resubmission according to book tags and product categories, respectively. Finally, a bottom fragment F8$_{history}$ carries a history of the recently viewed books.

Fragments F1$_{topresults}$, F2$_{allresults}$, F3$_{suggested1}$, F4$_{suggested2}$ and F8$_{history}$ of `search.dyn` that contain book listings link the users to `viewBook.dyn`, the most popular page of our store which generates and displays information on a specific book. The rest (F5$_{searchform}$, F6$_{resubmit1}$ and F7$_{resubmit2}$) link again back to the `search.dyn` page for repeating a search.

Template viewBook.html (V) The `viewBook.dyn` page synthesizes all information related to a book. The main fragment F1$_{bookinfo}$ displays its cover picture and summary info, while F2$_{bookreviews}$ shows reviews and ratings. F3$_{addtoshopbox}$ contains a panel for adding the book into a shopping box through various financial/shipping ways. Fragments F4$_{related1}$, F5$_{related2}$ and F6$_{related3}$ show listings for other related books (what other people bought, sponsored, used etc.) linking to again to `bookView.dyn`. The rest F7$_{repeatsearch1}$ and F8$_{repeatsearch2}$ link back to `search.dyn` for search resubmission according to book tags and product categories. Finally, fragment F9$_{searchform}$ contains the search form plus a few related suggested terms for repeating search.

Template shopBox.html (B) Fragment F3$_{addtoshopbox}$ of the previous template links to the shopping box of our application, implemented by the `shopBox.dyn` template. This template is called when a book or product is added to the shopping cart from the `viewBook.dyn` template. Fragment F1$_{checkout}$ of this template links to a less popular checkout page with various parameters that relate to shipping and payment. Fragments F2$_{related1}$, F3$_{related2}$ and F4$_{related3}$ show related book listings and link to `viewBook.dyn`. They also link to `shopBox.dyn` for

Stavros Papastavrou

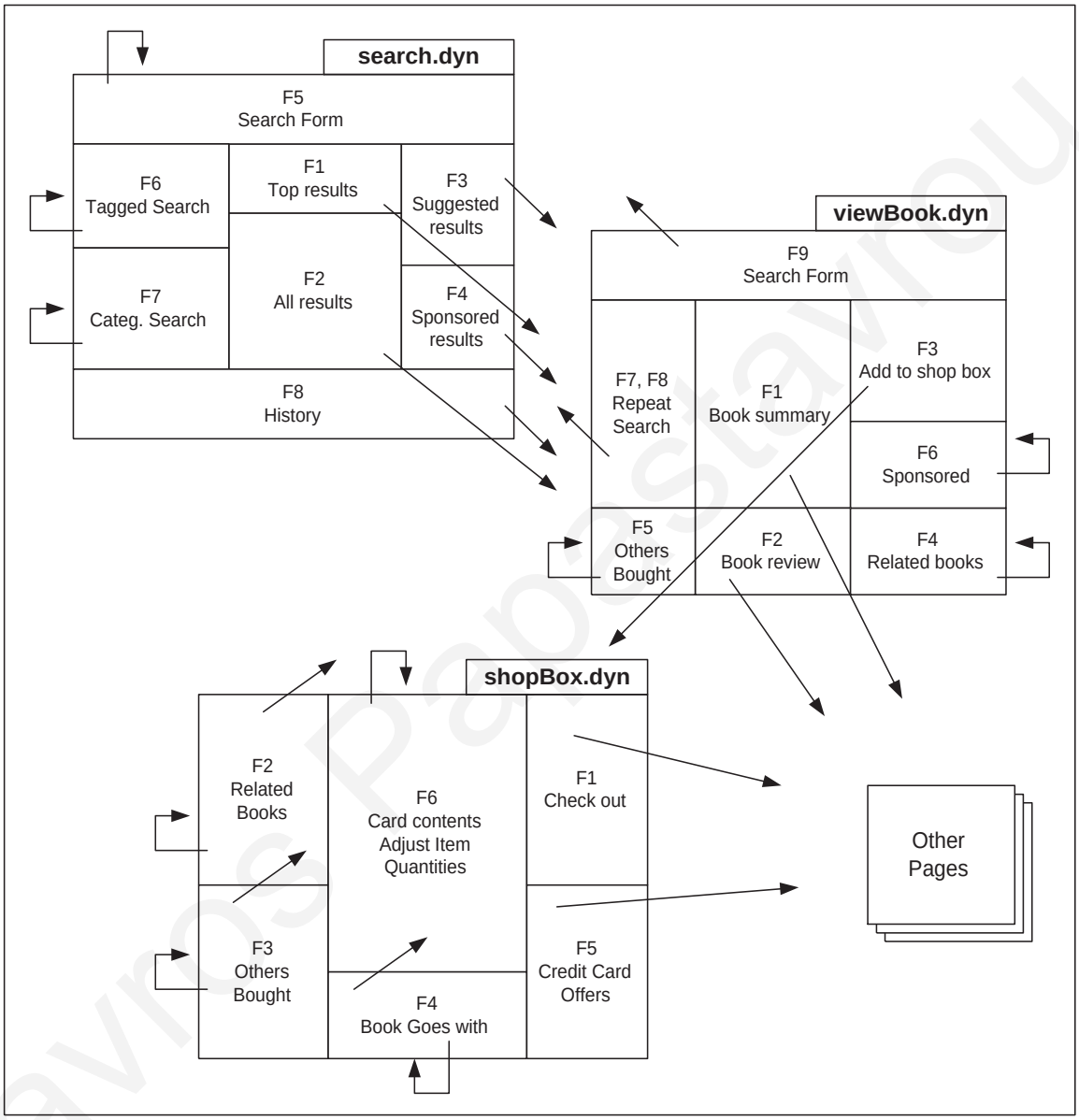

Figure 20: The Fragments of the three Most Popular Pages of the bookstore Application with their Linking Dependencies to other Templates

Stavros Papastavrou

adding the book directly into the shopping box without viewing it. F5$_{creditoffers}$ links to special credit card offerings. Finally, F6$_{adjustquantities}$ displays the shopping cart contents with options for increasing/decreasing quantities that link to the same page `shopBox.dyn`.

3.3 Current Approaches on Balancing QoS with QoD

3.3.1 General Framework

Current content materialization approaches for web database applications balance QoS and QoD by (a) increasing or decreasing the number of fragments that are materialized per template request and (b) by maximizing the QoD of the fragments that are materialized. Thus, the decision of which fragments are materialized is based on the importance of fragments. At design time, the web developer sets the importance of all fragments in every template according to the application semantics and requirements. For example, in our motivating application, fragments F1$_{topresults}$ and F2$_{allresults}$ that facilitate book searching in `search.dyn` may have greater importance weights compared to fragments F4$_{suggested2}$, F5$_{searchform}$ and F6$_{resubmit1}$ that facilitate suggested and sponsored results. The importance of an individual fragment is set only in the scope of its containing template with no consideration on the dependencies that may exist between that fragment and other content.

A popular methodology endorsed by related approaches is to define this template-wise importance of a fragment through a *QoD-specific importance weight* with values, for example, between zero and one. For instance, fragments F1$_{topresults}$ and F2$_{allresults}$ in `search.dyn` may have an importance weight value of 0.3 and 0.2, respectively, as opposed to fragments F4$_{suggested2}$,

Stavros Papastavrou

$F5_{searchform}$ and $F6_{resubmit1}$ that may have values of 0.1, 0.075 and 0.05, respectively. For practical reasons, such as maintaining a uniform way of fragment weighting, the sum of the importance values of all fragments inside a template sums up to one.

The users of a web database application can personalized their preferences on individual fragments by altering the QoD-specific weight values. This is facilitated with the use of user and session tracking techniques, such as session variables and cookies, as mentioned in Sections 1.3 and 3.1.

We illustrate the basic principle behind current approaches in Figure 21. In the example given, a particular user requests the template `search.dyn` for 12 consecutive times. At low workload, all fragments of the template are materialized (white in Figure 21) and returned fresh to the user. In this case, the instance of the dynamic page of the template is considered as "fully fresh". As workload increases, more fragments are reused from cache. The ones with the least QoD-specific important weight are the first to be reuse from cache. In our example in Figure 21, the weight of the fragments decreases from top to bottom and from right to left, with the fragment at the bottom right having the lower weight. At peak workload, the materialized fragments are even fewer than those reused from cache.

3.3.2 Related Approaches

The first study on balancing QoS-QoD for web database applications is found in [63]. This approach targets data-driven web applications with frequent data updates, such as stock-related information web sites. It compares three methodologies for enhancing QoS and QoD. The experimental findings reveal that server-side caching of materialized content at the web server is a better practice, than materializing content at the database server, or materializing content at all times without employing caching at all.

Stavros Papastavrou

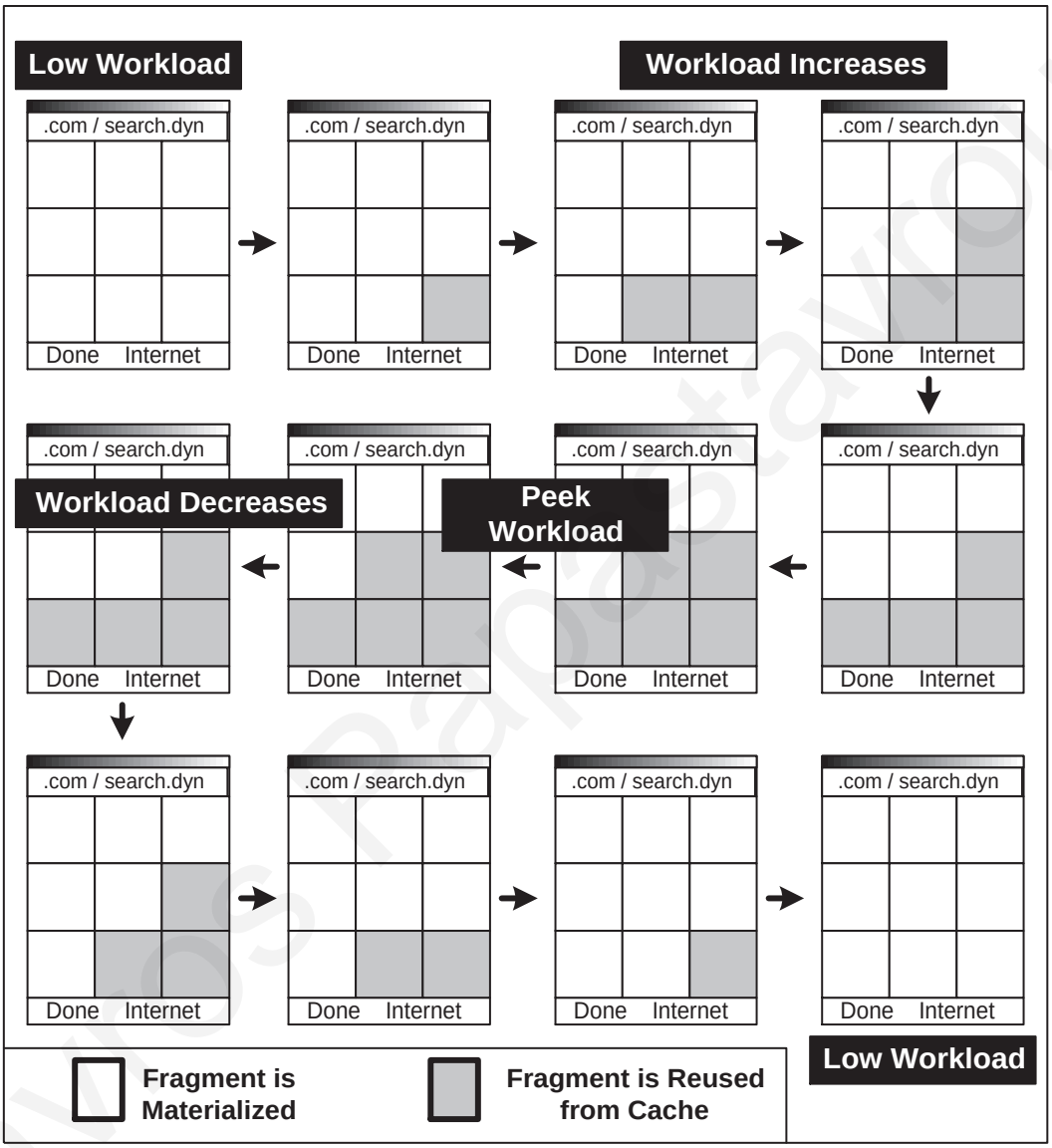

Figure 21: The basic principle behind current approaches. As workload increases, more fragments are reused from cache (dark). As workload decreases, more fragments are materialized (white).

Stavros Papastavrou

The proxy-based approach in [29], also suitable for data-driven web applications, assigns a user-specific importance weight to content fragments that are cached on proxy servers. The decision whether to materialize a fragment depends on user-profiled latency-recency metrics, calculated on demand at the proxy side. Latency refers to the end-to-end period required for downloading a data item. Recency refers to the number of updates on the data items [53]. In this approach, a dynamic web page is assembled at the proxy server, on user request, by reusing a variable number of cached fragments from the cache at the proxy and by fetching the rest fragments freshly materialized from the server side.

Extending [63], a more thorough QoS-QoD balancing approach for data-driven Wed applications is found in [65, 66]. According to this approach, the QoS-QoD balancing algorithms pre-materialize and store content fragments on the asynchronous cache between the application and the database server for immediate reuse (Figure 22). The QoD metric in this approach considers only two states of freshness for materialized content: “fresh” and “not fresh”. QoS is favored by the more eager materialization of fragments, according to which popular fragments are materialized as soon as they become invalid in the asynchronous cache. On the other hand, to favor QoD, fewer popular fragments are pre-materialized and cached. The approach secures first the QoD and then promotes QoS as much as possible by materializing the fragments that require the least system overhead. This approach is considered as *QoD-centric* in the sense that QoD thresholds must be met first.

Extending [66], the work in [97] balances QoD and QoS for data-driven web application systems using multi-faceted user-profiled Quality Contracts (QC). For this approach, the freshness metric considers the number of unapplied updates on data items, as well as QoS specific metrics such as system performance.

Stavros Papastavrou

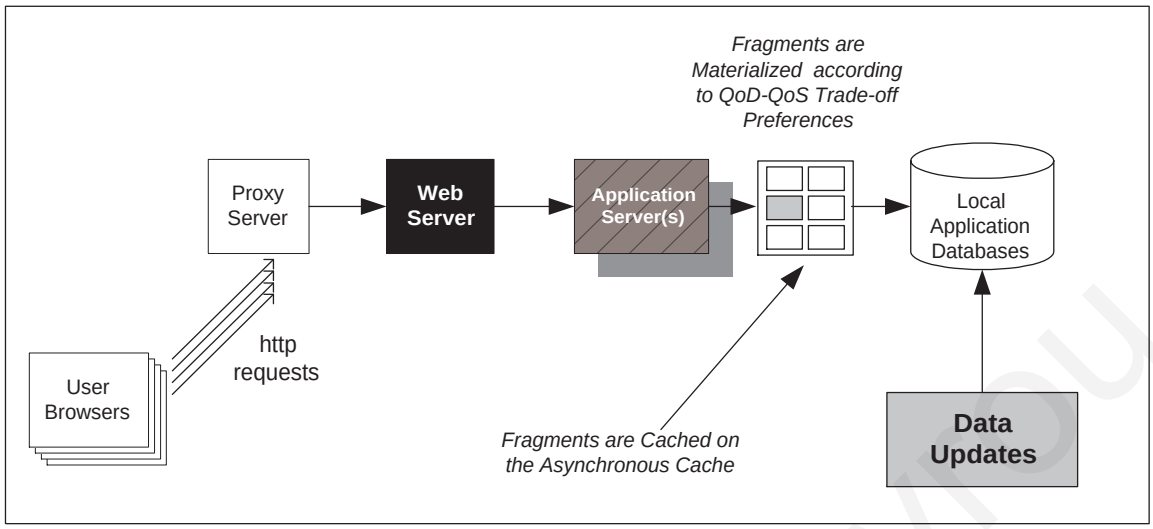

Figure 22: The QoS-QoD Tradeoff Approach for Balancing QoD with QoS. Pre-materialized content is stored on the asynchronous cache for immediate reuse.

The proxy-asynchronous-cache approach in [71], which targets data-driven web applications, shares characteristics of the approaches discussed above (Figure 23). More specifically, content fragments are materialized on the application server and pushed to the proxy servers, instead of being stored on an asynchronous cache, as employed by [71]. Pre-materialized content is "pushed" to the proxy only at periods when it is more likely to be requested. In other words, this approach attempts to synchronize content materialization with the user request frequency.

3.3.3 Other Related Approaches

In other related pull-based approaches, the proxy and the user sides are responsible for setting the policy and schedule on which fresh content is fetched (pulled) from the server side. In the simplest case, the freshness criteria in [43] are based on time-to-live (TTL) information. The approach in [38] suggests the use of sampling methods on a subset of the cached documents that determine the relative ratio between outdated and cached documents. Work in [37] suggests the

Stavros Papastavrou

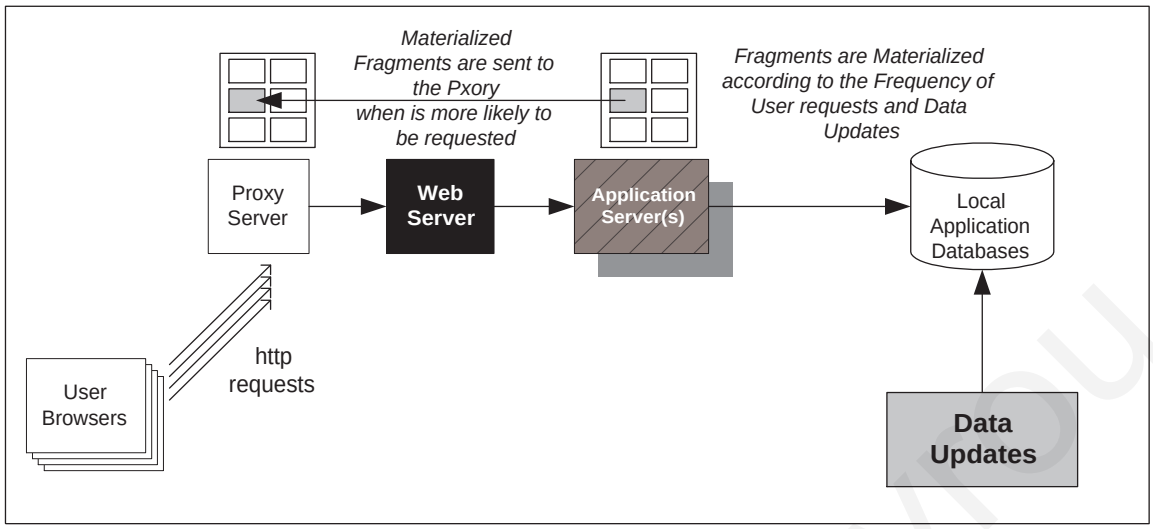

Figure 23: The Proxy-Asynchronous-Cache Hybrid Approach for Balancing QoS with QoD. Pre-materialized content is pushed to the proxy server.

use of probabilistic methods to estimate the change frequency of cached content using Poison processes.

The pull-based approach in [32] stresses the importance of popularity and content size for content pulling, in order to maximize the freshness of cached content. This is analogous to content multicasting [18], discussed in the previous chapter, according to which the most popular documents are those that are multicasted more often. In addition, [32] introduces the notion of "Perceived Freshness" that factors in the popularity of content objects towards an average perceived freshness.

Finally, balancing of QoS with QoD is also studied in the context of real time database systems (RTDBMS) [26] and on-line analytical processing (OLAP) [101]. To the best of our knowledge, there is no related provision for content dependencies in any of the above approaches.

Stavros Papastavrou

3.4 Current Shortcomings

A major drawback of the methodologies described above is that they all employ a flat, template-wise, QoD-specific importance weight on the fragments of a template, with no consideration for the overall semantics of the application. Consequently, this syntactic-based usage of QoD metrics employed in each case cannot capture content dependencies and user access patterns. Below, we define content dependencies, use our motivating application to give examples of current shortcomings and explain their complications. We also discuss the problems arise from not considering the user request patterns and look into possibilities of exploiting them.

- **No Provision for Link Dependencies**:

  Definition 3. (Link Dependency) *If there is at least one URL link or HTML Form inside fragment F_{source} that links to a template T_{dest}, then fragment F_{source} is link-dependent to template T_{dest}.*

  In other words, a source template links to a destination template with at least one link from within one of its fragments, providing the user with the means of navigating from the source template to the destination template. Links in the source template can be scattered in more than one of its fragments.

  For example, as illustrated in Figure 20, template `search.dyn` links to `viewBook.dyn` from within 5 fragments ($F1_{topresults}$, $F2_{allresults}$, $F3_{suggested1}$, $F4_{suggested2}$ and $F8_{history}$). We refer to those links as *cross-links*. Besides cross-links, a page may link to itself using *self-links*. The `shopBox.dyn` template self-links to itself from within four fragments

Stavros Papastavrou

(F2$_{bookreviews}$, F3$_{addtoshopbox}$, F4$_{related1}$ and F6$_{related3}$) for adding sponsored and suggested books directly into the shopping box, or for updating the quantity of books already in the shopping box.

The problem with link dependencies arises when a fragment that is link-dependent to the next template that a user will request is reused form cache due to increase server workload. For example, in template `viewBook.dyn`, if fragment F3$_{addtoshopbox}$ is reused from cache by the traditional QoS-QoD balancing algorithm, then it may not contain the correct HTML links so that the user can navigate to the `shopBox.dyn` page and add that particular viewed book in the cart.

In this case, the absence of a freshly materialized F3$_{addtoshopbox}$ (an unsatisfied link dependency) stalls the user session, which can only resume as soon as a fresh F3$_{addtoshopbox}$ is materialized and fetched to substitute the cached version at the user's browser. This is an instance of the generally known problem of "dangling's pointers of reference" in computer science, according to which a pointer variable does not point to a valid object of the appropriate data type.

- **No Provision for Set-View Dependencies**:

Definition 4. (Set-View Dependency) *A fragment F_1 in template T_x is set-view dependent to fragment F_2 in the same template T_x if both fragments must present consistent-to-each-other information.*

In other words, set-view dependencies relate to the necessity of having two content fragments synchronized in order to simultaneously present consistent information on the user's browser. This occurs when both fragments are materialized in the same request or by reusing

Stavros Papastavrou

synchronized cached versions. In the scope of this dissertation, we consider and provide support for both cases.

The problem with set-view dependencies arises when one of the two set-view dependent fragments is reused from cache, or both are reused from cache with asynchronous versions (an unsatisfied set-view dependency). For example, in template `shopBox.dyn`, if fragment $F4_{related3}$ is reused from cache, then it may show inconsistent results with the fresh fragment $F6_{adjustquantities}$, since it may suggest the purchase of a book that already exists in the cart of the user.

- **Generation Dependencies**:

Definition 5. (Generation Dependency) *A fragment F_{dest} in template T_x is generation-dependent to fragment F_{source} in the same template T_x if it relies on the processing outcome of F_{source} to materialize.*

In other words, the generation-dependent fragment F_{dest} is data-dependent on fragment F_{source}. For example, in template `search.dyn`, fragments $F3_{suggested1}$ and $F4_{suggested2}$ are generation-dependent to fragments $F1_{topresults}$ and $F2_{allresults}$, respectively, since they use their processing outcome to generate related and sponsored book listings.

The problem with generation dependencies is that, if $F1_{topresults}$ is not materialized (since it is tagged as "less important") then $F3_{suggested1}$ cannot be materialized (an unsatisfied generation dependency). In the scope of this dissertation, we provide support for generation dependencies as introduced in [34] and studied more thoroughly in [90].

Stavros Papastavrou

- **No Provision for the User Request Patterns**:

  Apart from content dependencies, no approach on balancing QoS with QoD considers the temporal locality of requested templates and other related metrics. [109] shows that, in addition to static documents and media files, the popularity of dynamic pages (and consequently of templates) also obeys a zipf-like distribution. In other words, fewer templates account for more requests in a structured, almost predictable manner: the most popular template is accessed roughly at a rate of 50%, the second most popular at a rate of 25% and so on. It has also been shown that for dynamic web applications, a small set of templates (approximately four) account for almost 95% of the requests [12], where this set of templates is stable over time [84]. Similarly, [39, 82] refer to "mostly working" user sessions, in which users exhibit a very strong temporal locality in their request patters on a small set of documents.

  The implications of not considering the above findings may have a dramatic negative impact on QoD and the overall user experience, since the temporal locality in the request patterns is highly related to link dependencies. Under heavy workload, it is more appropriate to materialize fragments with link dependencies to the template that is more likely to be requested next by a user. If the QoS-QoD balancing algorithm fails to do so, then the user will be unable to navigate to the next template and the session will stall. In this case, the session can resume only when fresh content fragments are materialized and fetched containing up-to-date links.

3.5 The Need for Semantics-based Metrics

To address the above shortcomings, we propose that content dependencies be encapsulated by new semantics-based data quality metrics. More specifically,

Stavros Papastavrou

- Link dependencies must be incorporated in such a way so that the they facilitate the materialization of fragments that link to the next template in a user's request pattern. The surveyed QoD metrics (Section 2.2) that characterize the freshness of materialized content, as well as the way those metrics are employed by current approaches, do not provide related provision.

  In order to incorporate link dependencies into the materialization procedure, the current template-wise QoD-specific importance of a fragment must factor in the link dependencies. In this way, the current traditional flat popularity of a fragment becomes dynamic in the sense that its value depends on the next fragment in a user's request pattern.

- Set-view dependencies must be incorporated in such a way so that they facilitate the materialization or synchronization of set-view dependent fragments. In order to do so, an additional data quality metric that factors in set-view dependencies must be introduced.

  This new set-view-aware metric must be independent from other metrics. The benefit from decoupling the set-view-aware metric is that it can be used independently where needed such as in a web database application with no link dependencies.

- Generation dependencies must be considered in the process of content materialization. Otherwise, generation-dependent fragments that are not materialized may limit the handling of link and set-view dependencies.

3.6 Chapter Summary

This chapter examined related work on balancing QoS with QoD and identified current shortcomings through extended discussion on our motivating application. The shortcomings is a result of dependencies between content fragments and templates. Failure to consider link, set-view and

Stavros Papastavrou

generation dependencies, as well the user request patterns, generates an array problems listed below:

Firstly, in case link dependencies are ignored, a user session might stall due to the absence of freshly materialized fragments containing valid links to the next template in the user's request pattern. When this occurs, a user should request fresh versions of a particular fragment, in order to continue navigation. The knowledge of the next template of the user can be facilitated by considering the users' request patterns, which exhibit increased similarity and locality.

Secondly, in case set-view dependencies are not considered, it is possible that fragments with related content might be served to the users unsynchronized, that is containing unrelated content. This occurs when at least one of the dependent fragments is not materialized.

Thirdly, if generation dependencies are not considered, it is possible that generation-dependent fragments that are not materialized may limit the processing of link and set-view dependencies.

In conclusion, new data quality metrics that incorporate content dependencies must be integrated in balancing QoS with QoD. Finally, user access patterns must be considered so the temporal locality of user requests facilitates link dependencies.

Stavros Papastavrou

Chapter 4

The key Components that Characterize our Approach

This chapter presents the conceptual framework of our approach for balancing QoS with data quality. It is based on three key components that measure data quality, exploit user access patterns and control QoS. All three key components are used by the materialization algorithms that are presented in the next chapter.

4.1 The Conceptual Framework

Current QoS-QoD balancing approaches fail to capture content dependencies and user access patterns thus leading to an array of problems, as detailed in Sections 3.4 and 3.5. To address these shortcomings, our approach (a) introduces novel data quality metrics that are sensitive to content dependencies and measure data quality, (b) introduces a custom scheme for handling and exploiting user access patterns and (c) employs a QoS-centric methodology for controlling QoS.

The three elements above are the three key components of our approach and are implemented by three corresponding modules at the application server layer in the general system architecture (Figure 24). In this way, applying our approach requires only modifications at the application

Stavros Papastavrou

server layer which is similar to installing any other popular application server such as ColdFusion, ASP.NET or the PHP application server.

This chapter presents in depth the three key components using examples from our motivating application. As we specify in the next chapter, the key components are used by the materialization algorithms either as functions calls or auxiliary programs. In brief, the key components are as follows:

The novel QoL and QoSV Metrics The first key component of our approach measures data quality through two semantics-based metrics that capture content dependencies. These metrics quantify the degree to which link and set-view dependencies are satisfied when a template is materialized to create a dynamic web page.

The first metric, called *Quality of Link* (QoL), is sensitive to link dependencies and measures the navigation ability of a user between consecutive templates in his/her template request sequence. QoL is analogous to maximizing QoD in a template so that the user is served with fresher content. However, the maximization of QoL increases the possibility of serving the user with fresh content that contains valid links to the next template to be requested. The intuition is that, fragments with link dependencies to the template that is more likely to be requested next must be preferred for materialization.

The second metric, called *Quality of Set-View* (QoSV), is sensitive to set-view dependencies. It measures the set-wise consistency of content served by considering the set-view dependent fragments in a template. Similar to maximizing QoD for freshness, the maximization of QoSV in a template increases the possibility of serving the user with set-view dependent fragments synchronized, that is containing consistent information.

Stavros Papastavrou

Both QoL and QoSV are employed by the *Materialization Algorithms Module* (see Figure 24) as we discuss in detail in Sections 5.1.3 and 5.2.

Usage Plans Our second key component, called *Usage Plans*, is a method for capturing the highly recurrent user behavior in its finest grain for web database applications such as our bookstore. The employment of Usage Plans is essential to the materialization algorithms under heavy workload. This is because, Usage Plans provide the means to the materialization algorithms to focus on the right set of fragments to materialize per template request. In this way, data quality is ensured while computational resources are spared for promoting QoS. Usage Plans are implemented and exploited by the *Speculation Profiling Module* responsible for speculating on the next template to be requested by a particular user.

The novelty of Usage Plans lies on the way recurrent user access patterns are captured and modelled into successive looping transitions between templates such as (a) consecutive requests for the same template and (b) consecutive requests back-and-forth between two specific templates. By employing Usage Plans, every user session is encoded on-the-fly into successive patterns of requests which are then exploited by a simple profile-based speculation mechanism.

QoS-centric Control Scheme The third key component of our approach, called the *QoS-centric Control Scheme*, regulates QoS defined in terms of server-perceived response time and is implemented at the *QoS Module*. The employment of the QoS-centric Control Scheme is essential to the materialization algorithms since its procedures and directions instruct the algorithms to define the QoS of users.

In brief, QoS is regulated by varying the maximum number of fragments to be materialized per template request. The goal is to materialize and serve fresh as many fragments as possible, given

Stavros Papastavrou

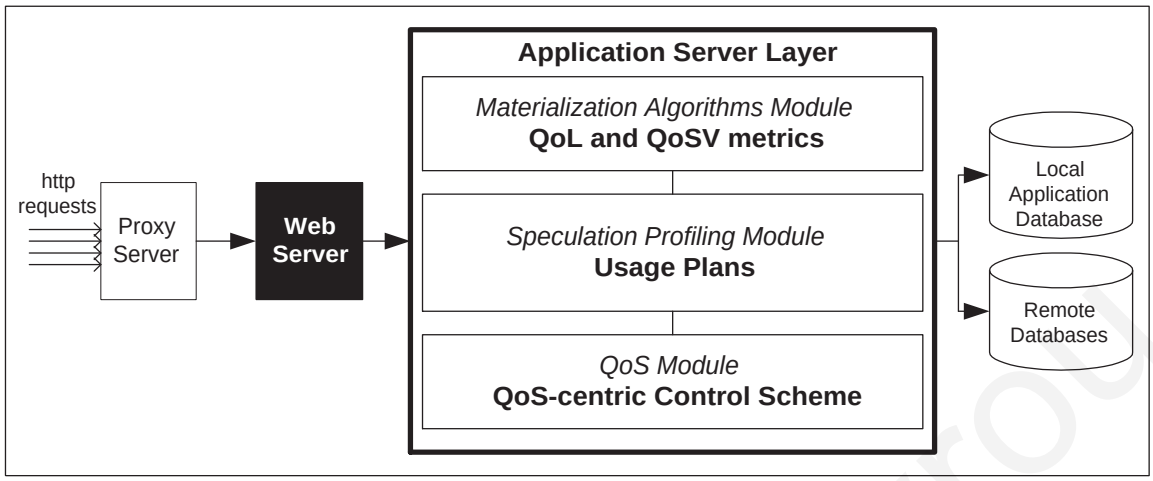

Figure 24: An Overview of the System Architecture of our Approach. The three key components are part of three corresponding modules implemented at the application server layer.

the current server workload and the maximum tolerable response time. Non-materialized fragments are reused from cache, sparing resources for promoting QoS and, consequently, affecting data quality in terms of QoL and QoSV.

4.2 New Data Quality Metrics: QoL and QoSV

4.2.1 Overview

The first key component in our approach measures data quality using two semantics-based metrics, namely QoL and QoSV, that substitute the traditional QoD metric. As detailed in Chapter 3, current QoS-QoD approaches (namely, traditional QoD approaches) handle the importance of a fragment as a flat metric that captures a static, user-specific preference to a fragment. This importance is expressed through a QoD-specific importance weight. Moreover, the overall QoD of a template that generates a dynamic web page is the sum of the QoD-specific importance weights of the freshly materialized fragments in that page.

Stavros Papastavrou

Definition 6. (Traditional QoD Approach) *The content materialization approach according to which, the fragments that are selected for materialization in a template are those with the highest QoD-specific importance metric is called Traditional QoD Approach.*

To formally define QoD, we first define the concept of QoD importance weight of a fragment.

Definition 7. (weightQoD(F,T)) *weightQoD(F,T) is a function that returns the QoD importance factor of fragment F in template T. Let F_1, F_2, ..., F_n be the member fragments of template T. The sum of the QoD importance weights of all fragments in T is 1:*

$$\sum_{i}^{n} weightL(F_i, T) = 1$$

As discussed in the previous chapter, the QoD importance weights of fragments are typically expressed with a decimal value between 0 and 1, whereas all QoD importance weights of fragments in a template sum up to 1.

Definition 8. (contrQoD(F,T)) *contrQoD(F,T) is a function that returns the QoD contribution of fragment F in template T when that template is requested by a particular user. If fragment F is materialized on this request, then its QoD contribution is 1. If fragment F is reused from cache, then its linking contribution is 0:*

$$contrQoD(F,T) = \begin{cases} 1 & \text{if F is materialized in T} \\ 0 & \text{if F is reused from cache in T.} \end{cases}$$

Stavros Papastavrou

Definition 9. (QoD(T)) *QoD(T) is a function that returns the QoD of template T. Let F_1, F_2, ..., F_n be the member fragments of template T. The QoD(T) of template T is defined as follows:*

$$\text{QoD(T)} = \sum_{i}^{n} weightQoD(F_i, T) \times contrQoD(F_i, T)$$

In other words, if all fragments inside template T are materialized when T is requested by a user, then the QoD for template T has the maximum value of 1. If a particular fragment in T is not materialized, then the QoD value is reduced according to the QoD importance weight of that fragment.

In our approach, however, we consider the importance of a fragment to be a multi-faceted metric which relates (a) to the ability of a user to request the next template and (b) to the relation of that fragment to other fragments inside its template. Subsequently, we assign two types of importance weights to a fragment: one that captures its link dependencies and one that relates to its set-view dependencies.

Figure 25 compares side by side our new metrics in respect to the traditional QoD metric. On the left, the traditional QoD approach assigns individual importance weights to fragments as discussed in Section 3.3. In our approach on the right, as we elaborate below, QoL importance weights are assigned from each fragment toward each link-dependent template and QoSV importance weights are assigned between set-view dependent fragments in the same template.

Stavros Papastavrou

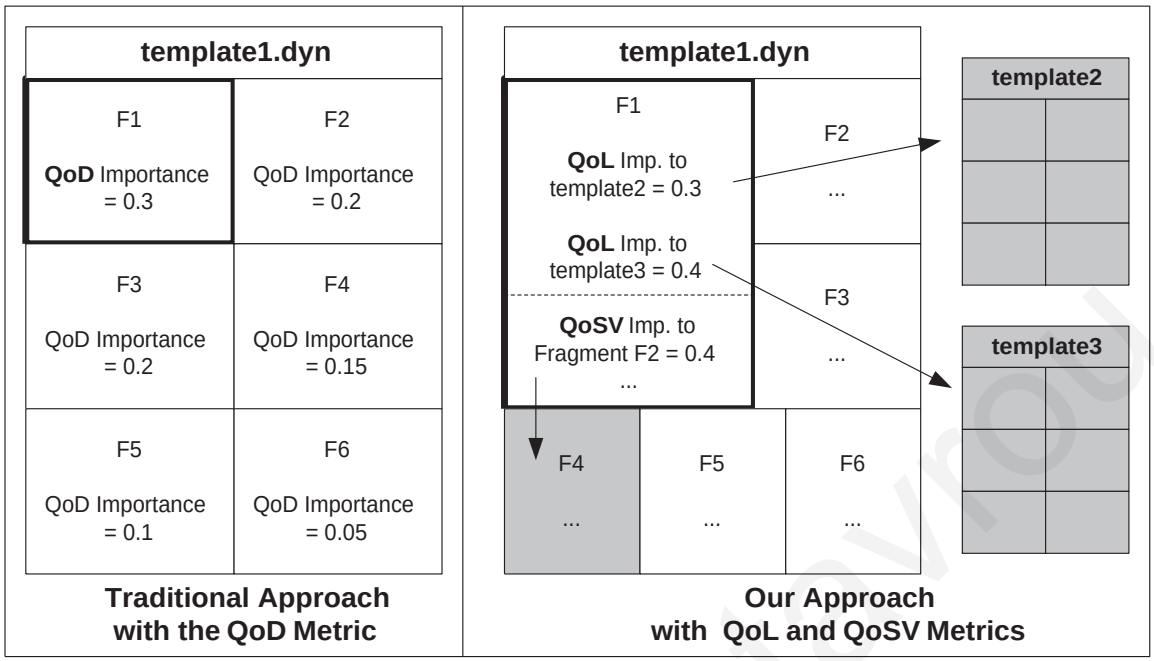

Figure 25: Comparison of our novel Data Quality Metrics with the Traditional QoD Metric

4.2.2 Quality of Link (QoL)

The metric of QoL quantifies the existence of freshly materialized fragments inside a template T_s with link dependencies toward a target template T_d. To formally define QoL, we first define the concept of QoL importance weight of a fragment toward a link-dependent template.

Definition 10. (weightL(F_i,T_s,T_d)) *weightL(F_i,T_s,T_d) is a function that returns the QoL importance factor of fragment F_i in template T_s toward template T_d. Let F_1, F_2, ..., F_n be the member fragments of template T_s. If there is at least one fragment in template T_s with a link dependency to template T_d, then the sum of the QoL importance weights of all fragments in T_s toward T_d is 1:*

$$\sum_{i}^{n} weightL(F_i, T_s, T_d) = 1$$

In other words, weightL(F_i,T_s,T_d) measures the navigation/linking importance of fragment F_i in template T_s toward template T_d. In this way, the importance of F_i is not static as this is the case

Stavros Papastavrou

with the traditional QoD approach. Instead, its dynamic since it depends on a target template T_d. Moreover, the QoL importance weights of fragments are expressed with a decimal value between 0 and 1, whereas all QoL importance weights of fragments in a template toward a specific template sum up to 1.

We demonstrate this with an example from our motivating application. At a specific point, our user is viewing a book through template `viewBook.dyn`. If the expected behavior of our user is to add the book in the shopping cart by linking to `shobBox.dyn`, then the importance of fragment $F3_{addtoshopbox}$ in template `viewBook.dyn` which links the user to `shopBox.dyn` is relatively higher to other fragments that link the user to other templates.

Consider an alternative scenario where the expected behavior of our user, after viewing the book in `viewBook.dyn`, is to repeat search using suggested book listings. In this case, the importance of fragments $F7_{repeatsearch1}$ and $F8_{repeatsearch2}$ that perform tagged book search by linking to template `search.dyn` is relatively higher to the importance of fragment $F3_{addtoshopbox}$. Moreover, fragment $F9_{searchform}$ that also links to template `search.dyn` has higher importance than $F3_{addtoshopbox}$.

Definition 11. (contrL(F_i,T_s)) *contrL(F_i,T_s) is a function that returns the linking contribution of fragment F_i in template T_s when that template is requested by a particular user. If fragment F_i is materialized on this request, then its linking contribution is 1. If fragment F_i is reused from cache, then its linking contribution is 0:*

$$contrL(F_i, T_s) = \begin{cases} 1 & \text{if } F_i \text{ is materialized in } T_s \\ 0 & \text{if } F_i \text{ is reused from cache in } T_s. \end{cases}$$

Stavros Papastavrou

Definition 12. (QoL(T_s,T_d)) *QoL(T_s,T_d) is a function that returns the QoL of template T_s toward template T_d. Let F_1, F_2, ..., F_n be the member fragments of template T_s. The QoL(T_s,T_d) of T_s toward T_d is defined as follows:*

$$\text{QoL}(\text{T}_s,\text{T}_d) = \sum_{i}^{n} weightL(F_i, T_s, T_d) \times contrL(F_i, T_s)$$

In other words, if all fragments inside template T_s with link dependencies to T_d are materialized when T_s is requested by a user, then the QoL for template T_s toward T_d has the maximum value of 1. If a particular linking fragment from T_s toward T_d is not materialized, then the QoL value is reduced according to the QoL importance weight of that fragment toward T_d.

Recall our previous example where the expected behavior of our user, after viewing a book in template `viewBook.dyn`, was to repeat search by navigating to `search.dyn` using suggested book listings. In this case, if fragments $F7_{repeatsearch1}$, $F8_{repeatsearch2}$ and $F9_{searchform}$ in `viewBook.dyn` are materialized, then the QoL `viewBook.dyn` toward `search.dyn` is maximized. If one of these three fragments is not materialized, then the QoL of the page is reduced according to the individual QoL importance of that fragment toward template `search.dyn`. If none of these three fragments is materialized, then the QoL of the page is zero, implying that the user has no means of navigation to template `search.dyn`, since no fragment contains fresh links to that template.

4.2.3 Quality of Set-view (QoSV)

The metric of QoSV quantifies the overall set-wise consistency of set-view dependent fragments inside a template. Similar to QoL, in order to define QoSV, we first define two auxiliary

Stavros Papastavrou

functions that capture the QoSV importance weight of a fragment. We express the QoSV importance weight of two set-view dependent fragments in a similar manner to QoL, however, the QoSV importance weight is not dynamic since it does not additionally depend on other templates.

Definition 13. (weightSV($\mathbf{F}_i$,$\mathbf{F}_j$,T)) *weightSV(F_i,F_j,T) is a function that returns the QoSV importance weight between fragments F_i and F_j in template T. Let F_1, F_2, ..., F_n be the member fragments of template T. If there is at least one set-view dependency in template T, then the sum of all the QoSV importance weights in T is 1:*

$$\sum_{i,j}^{n} weightSV(F_i, F_j, T) = 1$$

In other words, weightSV(F_i,F_j,T) measures the importance of having fragments F_i and F_j in template T synchronized/present consistent content. The QoSV importance between two fragment is set according to the application semantics, as discussed in Section 3.3 using a decimal value between 0 and 1. In addition, all the QoSV importance weights in a particular template must sum up to 1.

Definition 14. (contrSV($\mathbf{F}_i$,$\mathbf{F}_j$,T)) contrSV(F_i,F_j,T) *is a function that returns the set-view contribution of fragments F_i and F_j in template T, when that template is requested by a particular user. If fragments F_i and F_j are both materialized on this request, then their linking contribution is 1. If fragments F_i and F_j are both used from cache with synchronous versions (with the same timestamp), then their linking contribution is also 1. In any other case, their linking contribution is 0:*

Stavros Papastavrou

$$contrSV(F_i, F_j, T) = \begin{cases} 1 & \text{if both } F_i \text{ and } F_j \text{ are materialized in T} \\ 1 & \text{if both } F_i \text{ and } F_j \text{ are reused form cache with the same timestamp} \\ 0 & \text{otherwise.} \end{cases}$$

Definition 15. (QoSV(T)) *QoSV(T) is a function that returns the QoSV of template T. Let F_1, F_2, ..., F_n be the member fragments of template T_s. The QoSV(T) is defined as follows:*

$$\text{QoSV(T)} = \sum_{i,j}^{n} weightSV(F_i, F_j, T) \times contrSV(F_i, F_j, T)$$

In other words, if all pairs of set-view dependent fragments in a template are synchronized when T is requested by a user, then the template T is fully set-view consistent. If one pair of set-view dependent fragments is not synchronized, then the overall set-view consistency of their template is reduced according to the QoSV importance of that particular set-view dependency. Our approach supports only pair-wise definition of set-view dependencies, with not support for recursive definitions that define a set-view dependency between three or more fragments.

4.3 Usage Plans and Profile-based Speculation

This section introduces and explores in depth the notion of Usage Plans (UPs), which are the second key component of our approach. Usage Plans are the driving force behind the Speculation Profiling Module responsible for speculating on the next template to be requested by a particular user, also presented in detail in this section.

Stavros Papastavrou

4.3.1 Overview

Currently, a mechanism for correctly speculating on the next template to be requested by a user is not available. Under heavy workload, such a mechanism would enable a QoS-QoD balancing server to select the right set of fragments in a template to materialize that contain links to that next template. As a result, the materialization of "not important at the moment" fragments is avoided and resources are spared for promoting QoS.

There are a number of data mining and machine learning approaches in the context of *Web Usage Mining*, which attempt to identify access patterns on objects (such as web pages or data items) in order to make them more readily available to users over the web [106]. These approaches operate on a large number of objects and require the progressive building and refinement of access models for that objects. In our context, however, we focus on a well-defined model that consists of a small number of popular templates derived from the conceptual design of web database applications.

The intuition behind the second key component of our approach is that a user session in a modern e-commerce web application, as discussed in Section 3.4, is a series of consecutive requests for a small, stable-over-time, set of templates. In this series, recurrent access patterns such as requests for the same template or looping back-and-forth requests between two particular templates are typical.

UPs capture the above recurrent access patterns. If put together, all the UPs of an application comprise a finite state machine which provides a well-defined, model-driven representation of user navigation in a web database application.

To build our speculation module, we make use of this well-defined model with a simple, profile-based feedback mechanism. At run time, this mechanism encodes a user's behavior in

Stavros Papastavrou

terms of UP switching. Based on that behavior, it speculates on the next template to be requested by a user. Finally, for feedback purposes, it compares its speculation with the actual user behavior in order improve future speculations.

4.3.2 Usage Plans

4.3.2.1 Template Transitions

As mentioned above, a user session is a series of consecutive template requests. Navigation to a new dynamic web page instance of the same template is through the self-links that are defined in the template. Cross-links in templates are used for navigating to a dynamic web page instance of a different template. As presented below in Example 1, the user session begins by performing a couple of book searches using the `search.dyn` template (S). Subsequently, the user picks a book for viewing from the results by navigating to `viewBook.dyn` (V). Following that, the user picks a second book for viewing from a related book listing inside V. The book is added to her shopping box by navigating to `shopBox.dyn` (B) and so on. Thus, the user session of Example 1 is reflected by the series of template transitions:

Example 1: S→S→V→V→B→V→B→...

It should be noted that a user session represented as a series of transitions may be different at the user and at the server side. In particular, the "back" button is not considered to be a transition at the server side since it displays a previously materialized page, sitting in the cache of the web browser at the user side and is not visible to the application server. In Example 1, a back button request could be possibly issued by the user after the first request for template V. That would display to the user the previously materialized search page from where a different book for viewing

Stavros Papastavrou

could be picked. This case is reflected below in Example 2 after the first occurrence of template V. The transition that is only visible to the user side is shown in parenthesis:

Example 2: S→S→V(→S)→V→B→V→B→...

4.3.2.2 Types of Usage Plans

All possible transitions in a user session of a web database application can be represented by a well-defined model, namely the UP-FSM (FSM stands for Finite State Machine), whose states relate to the templates of the application and transitions relate to the navigation between the templates.

Definition 16. (UP-FSM) *The UP-FSM is a finite state machine whose states are the template of the web database applications and the transitions are the linking alternatives between the templates.*

Definition 17. (Usage Plans) *A single-hop transition from a template to itself and a double-hop transition back-and-forth between two templates are called Usage Plans. The former is called uni-Usage Plan (uni-UP) and the latter is called bi-Usage Plan (bi-UP).*

Figure 26 presents five Usage Plans of the bookstore application on the UP-FSM with the three most popular templates of the application.

Two typical examples of uni-UPs in our motivating application are (a) consecutive user searches using the `search.dyn` template (denoted as S*) and (b) consecutive book views though the `viewBook.dyn` template (denoted as V*). Since uni-UPs are realized through self-links. The Example 3 below presents the uni-UP plan S* for consecutive transitions on template `search.dyn`.

Example3: Requests S→S→...→S are Usage Plan S*

Stavros Papastavrou

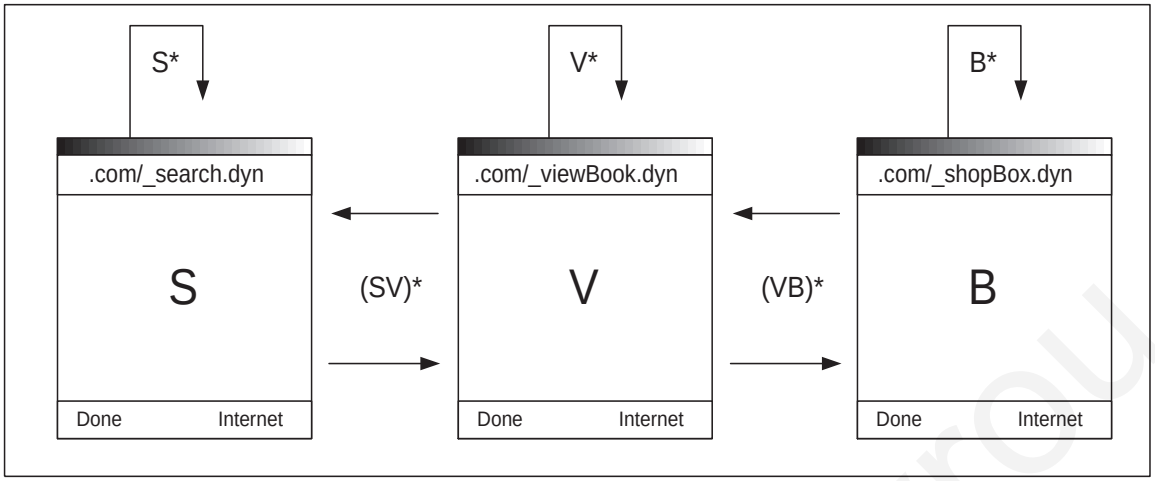

Figure 26: Five Usage Plans of the bookstore Application on the UP-FSM: Three uni-Usage Plans S*, V*, B* and two bi-Usage Plans (SV)* and (VB)*.

A typical example of a bi-UP is consecutive requests between `viewBook.dyn` and the `shopBox.dyn` templates (denoted by (VB)*) in the scenario where a user views a book, adds it in the buy box and then picks a suggested book to view from within the `buyBox.dyn`, adds it in the buy box and so on. The Example 4 below presents the bi-UP plan (VB)* that reflects the above scenario of template transitions.

Example 4: Pattern V→B→V→B→ ... V→B is Usage Plan (VB)*

4.3.2.3 Sequencing Usage Plans

Two Usage Plans cannot share the same cross-link or self-link. In other words, *every transition between two templates is a member of only one Usage Plan*. This statement is very important because it allows us to define a *session to consist solely of a sequence of Usage Plans*. We demonstrate this with an example user session of our motivating application, shown in Figure 27, in which the user performs first a search for a book three times in a row using S, views a couple of books using V and adds the last viewed book in the shopping basket using B. Then from within B, the user picks a suggested book to view using V and then adds it to the shopping basket using B.

Stavros Papastavrou

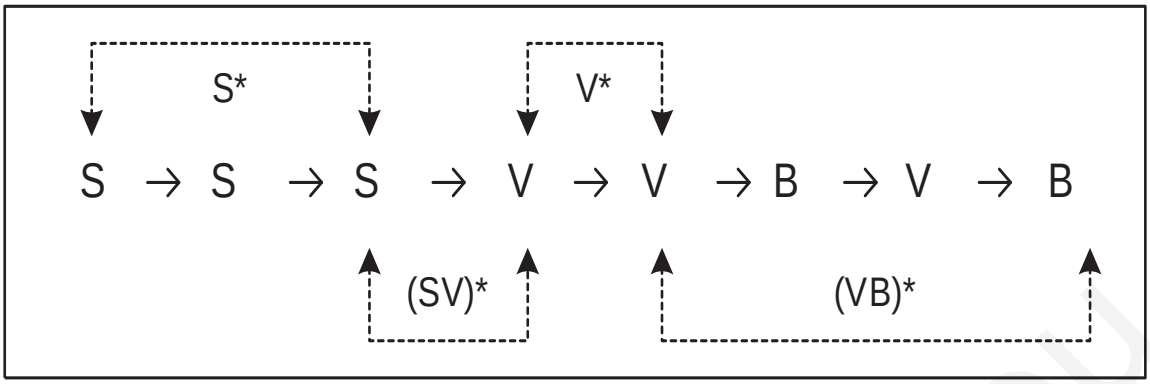

Figure 27: A Session Illustrated as a Sequence of Usage Plans. Note that each Usage Plan is immediately followed by another one. This is because every transition between two templates is a member of only one Usage Plan.

The sequence of templates is shown along with the projected Usage Plans that emerge. Note that Usage Plans do not overlap, since every transition between two templates is a member only of one Usage Plan.

In addition, it should be noted that the second Usage Plan in order in the example of Figure 27, namely, (SV)* does not include both of its template transitions, namely S→V and V→S. In this case, the Usage Plan was "partially completed" by the user but was still present in the sequence of Usage Plans.

4.3.3 Profile-based Speculation

4.3.3.1 Overview

As explained in Section 4.3, our profile-based speculation approach is not a contribution to web usage mining. As opposed to other approaches that operate on a large set of web documents, our approach operates only on a small set of templates that are captured by a well-defined model called the UP-FSM. As we discuss in the next chapter, our profile speculation is used as a function call by the materialization algorithms, so that it can be substituted in the future by other appropriate mining approaches.

Stavros Papastavrou

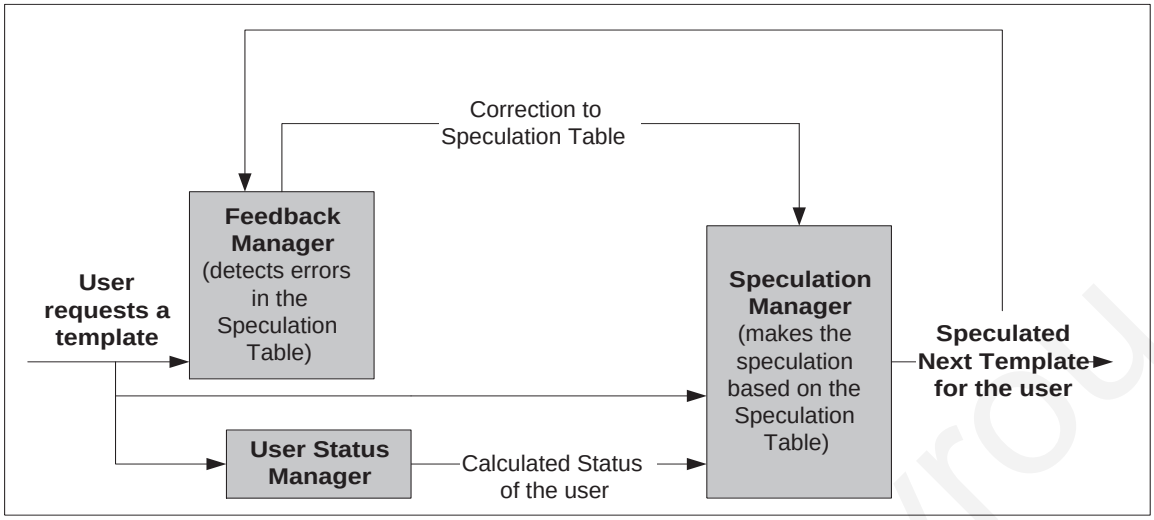

Figure 28: An Overview of the Profiling Mechanism.

The intuition behind our simple speculation mechanism is that users follow specific patterns when pursuing a specific task such as picking up a number of books to purchase. For example, "spontaneous" users may prefer to add books in their shopping basket immediately out of the lists of suggested as opposed to "hesitating" users who prefer to view them prior to adding them in their baskets. These two cases of behavior, among others, reflect well-known/predefined sequences of Usage Plans that can be exploited to reveal the next template that a user might request.

Our profile-based speculation works in three distinct steps:

- First, a user session (a sequence of template requests) is constantly monitored and encoded in terms of Usage Plan switching.
- Secondly, a speculation on the next template to be requested by the user is issued on every user request by considering prior user behavior on Usage Plan switching.
- Finally, for correcting/feedback purposes, the previous speculation for that user is compared to the actual requested template.

Stavros Papastavrou

Consequently, the profiling mechanism, illustrated in Figure 28, is a three-stage procedure that involves the following corresponding components: a *User Status Manager*, a *Speculation Manager* and a *Feedback Manager*. In the rest of this section, we examine how the User Status Manager models user behavior. Following that, we examine the speculation mechanism implemented by the Speculation Manager. Finally, we discuss how the Feedback Manager handles wrong speculations.

4.3.3.2 Preparing the Speculation: The User Status Manager

The first step in our profile-based speculation is a mechanism for identifying the current status of a user as this is shaped through the sequence of his/hers requests. This is handled by the User Status Manager and is triggered on every user request. The output is a description of the user status in terms of Usage Plans switching.

As mentioned earlier, the intuition behind our simple speculation mechanism is that every single user follows a pattern when pursuing a specific task. For "spontaneous" users, that immediately buy suggested books, the dominant Usage Plan in their session is B*. For "hesitating" users, that prefer to view suggested books prior to adding them in their baskets, the dominant Usage Plan is (VB)*. For "thorough" users, that prefer to search for every single book they add in their baskets, the dominant pattern is S→V→B. In addition, "searchy" users spend increased time on Usage Plan S* and, finally, "savvy" users spend time on the Usage Plan U* for viewing used books.

To model possible user behaviors, the User Status Manager employs a simple 3-state FSM, called the *User Status FSM* (US-FSM). Its states reveal the looping behavior of a particular user *only* in terms of switching between types of Usage Plans. In this way, the number of states on the US-FSM is much smaller, as opposed to having the Usage Plans themselves as the states. By

Stavros Papastavrou

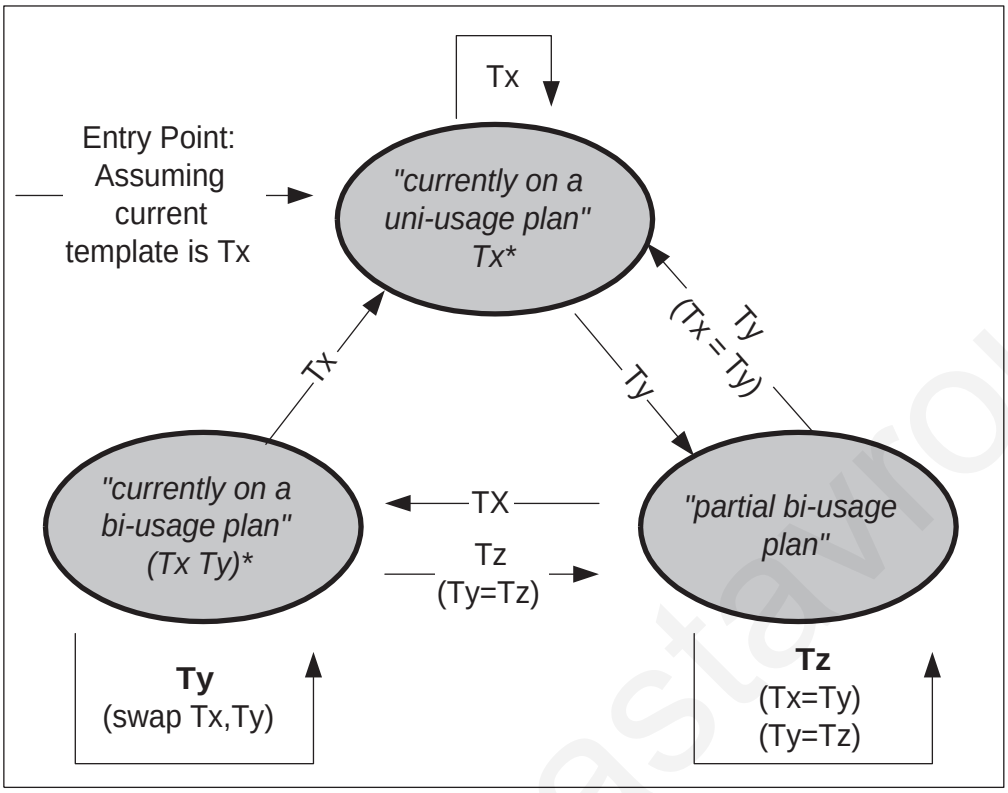

Figure 29: The User Status FSM for Determining the User Status.

making the US-FSM independent from templates allows for seamless addition of new, or deletion of existing templates. Illustrated in Figure 29, the states on the US-FSM are as follows:

- The first state, “currently on a uni-Usage Plan”, implies that the user has requested the same template again for at least one time (for example, S→S). In other words, it reveals that the user has begun or is currently looping on a template.

- The second state, “currently on a bi-Usage Plan”, implies that the user has at least looped once between the same two templates (for example, S→V→S).

- The third state, “partial bi-Usage Plan”, implies that the user has completed only one transition of any bi-Usage Plan. This occurs in three cases: (a) the user has just exited a uni-Usage Plan (for example, S→S→B), (b) the user has just exited a bi-Usage Plan (for example,

Stavros Papastavrou

S→V→S→V→B) and (c) the user has requested a different template for the third consecutive time (for example, S→V→B).

As mentioned earlier, the user status is independent from particular templates as it is only reveals the looping behavior of a user. Therefore, instead of using the complete list of available templates of a web database application as the inputs for the US-FSM, only three discrete templates are required (T_x, T_y and T_z) that denote any three different templates. To prepare for each request, the assignments in the parenthesis for every transition represent the required adjustments to the input values of T_x, T_y and T_z.

We demonstrate how the US-FSM works with an example of a user who jumps from a uni-UP to a bi-UP: Initially, every user state is set to "currently on a uni-Usage Plan" and the current template that a user is viewing is T_x (in our case, template `search.dyn`). Subsequently, if the user requests a different template T_y (for example, `viewBook.dyn`), then its state changes to "partial bi-Usage Plan" and no assignments are required. If the user requests back template T_x (`search.dyn`), then its state changes to "currently on a bi-Usage Plan" since it has completed a loop of the bi-UP (T_xT_y)* (in our case, (SV)*).

In a slight variation of the above scenario, the user requests template T_y (`viewBook.dyn`) for the second consecutive time, instead of requesting back template T_x (`search.dyn`). In this case, the user is entering the state "currently on a uni-Usage Plan" with the V* Usage Plan. To prepare the next user request, T_x becomes T_y.

4.3.3.3 Making the Speculation: The Speculation Manager

The second step in profile-based speculation, which is the actual speculation on the next template to be requested by a user, is handled by the Speculation Manager. Given the current status

Stavros Papastavrou

of a user, as this is extracted by the User Status Manager on every user request, the Speculation Manager speculates on the next template by considering patterns of Usage Plan switchings. These patterns are set a-priori, since the model of our application is well-defined. Nevertheless, as we explorer later, changes to those patterns due to changes in user behavior can be immediately applied.

The Speculation Manager lists all the patterns of Usage Plans switchings on a pattern-mapping structure called the **Speculation Table** (Figure 30). Since the speculation is a user-wise procedure, one speculation table is initialized and assigned to every new user session.

Each row on the Speculation Table represents a possible change in user behavior using a quadrable pattern and an output value. The quadrable pattern comprises of (a) the previous status of the user, (b) its current status (as returned by the User Status Manager), (c) the previous template that the user had requested and (d) the currently requested template of the user. Finally, each quadrable relates to a speculated Usage Plan that reveals the next template that the user will request.

For our example bookstore application and by excluding unnecessary rows due to non-existent links, all the possible combinations between pairs of user states (out of three) and pairs of templates (out of four) produce a 40-row speculation table. Figure 31 presents the pseudocode for constructing the Speculation Table.

At line 12 of the pseudocode, an initial Usage Plan that at least contains the target template must be selected. This selection can be based on the application requirements or any other profiling procedure that records the behavior of users and it is out of the scope of this dissertation. Nevertheless, as we discuss next, the Feedback Manager corrects wrong initial selections for speculated Usage Plans.

Stavros Papastavrou

Previous Status	Current Status	Current Template	Requested Template	Speculated Usage Plan
"Currently on a uni-UP"	"Partial uni-UP"	S	S	**S***
"Currently on a uni-UP"	"Partial uni-UP"	S	V	**V***
"Currently on a bi-UP"	"Currently on a bi-UP"	V	S	**(SV)***
...				
"Currently on a uni-UP"	"Currently on a uni-UP"	V	V	**V***
"Currently on a uni-UP"	"Partial uni-UP"	V	B	**(VB)***
"Currently on a uni-UP"	"Currently on a uni-UP"	B	B	**B***

Figure 30: The Speculation Table for the Bookstore Application (only 4 entries are shown).

Procedure Create Speculation Table

```
1  for each state st_prv  in the US-FSM
2
3     for each state st_nxt  in the US-FSM
4
5        for each template tmpl_src   in the application
6
7           for each template templ_dst   in the application
8
9              if transition (tmpl_src  → tmpl_dst)
10                is part of any usage plan   then
11                {
12                   UPx = select_a_UP(st_prv, st_nxt, tmpl_src, tmpl_dst)
13
14                   insert into Speculation table the tupple
15                            (st_prv, st_nxt, tmpl_src, tmpl_dst, UPx)
16                }
```

Figure 31: Pseudocode for Creating the Speculation Table

Stavros Papastavrou

We refer to the second row of the table, as displayed in Figure 30, to give an example of making a speculation: Consider a user who is performing book searches for a while via the `search.dyn` template S. Suddenly, our user decides to view a book from the search results and therefore requests the `viewBook.dyn` template V. Translated into Usage Plans switching, the user status is changing from "currently on a uni-Usage Plan" to "partial bi-Usage Plan". In this case, the Speculation Table suggests that our user is about to enter the V* Usage Plan and therefore, the next template to be requested is V*.

4.3.3.4 Correcting the Speculation: The Feedback Manager

The third and final module in profile-based speculation, handled by the Feedback Manager, is a procedure for comparing the actual user behavior against the Speculation Table towards making a correction. In this way, the profile mechanism stores knowledge on a user's preferable sequence of Usage Plans and adapts to changes in the user's "typical behavior". Below, we demonstrate this procedure using two examples from our application.

According to the first example shown in Figure 32, our user performs a couple of book searches (S→S) and subsequently picks up a book for viewing by requesting template V. The User Status Manager records the change in the status of the user and, subsequently, the Speculation Manager returns the Usage Plan V*. However, the user next requests template B instead of the anticipated template V, which is consistent to Usage Plan V* in the Speculation Table. In response, the Feedback Manager detects the error on the previous speculation and updates the last column of the Speculation Table to (VB)*, since the transition V→B is part of the Usage Plan (VB)*.

In the second example shown in Figure 33, our user first performs one search and then picks up a book for viewing. This is repeated once more and subsequently the user adds the book into his shopping box (B). However, the user requests template B instead of the anticipated template V

Stavros Papastavrou

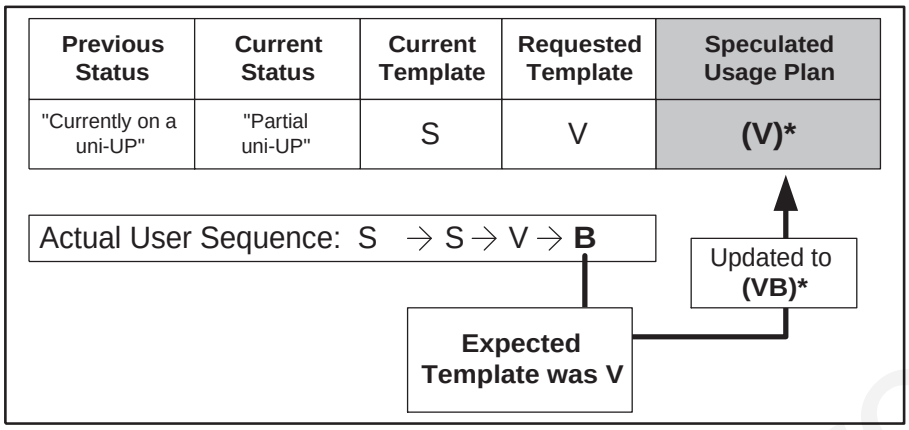

Figure 32: First Example of Feedback on the Speculation Table for the bookstore Application.

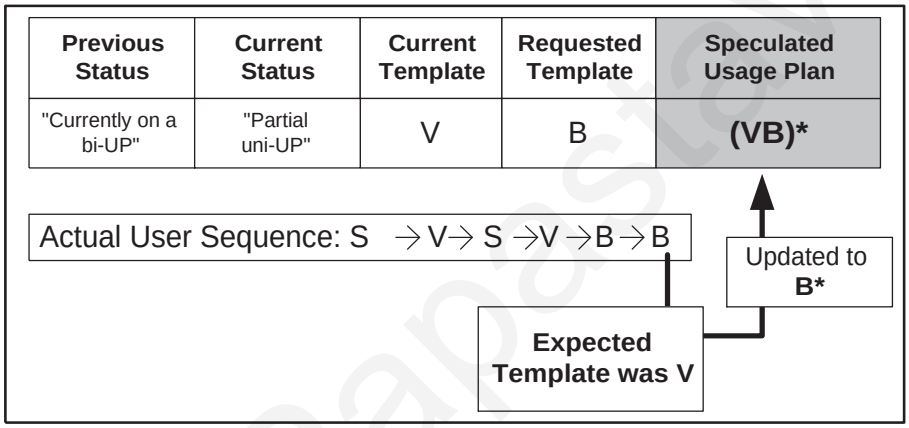

Figure 33: Second Example of Feedback on the Speculation Table for the bookstore Application.

that is consistent to Usage Plan (VB)*. Similarly to the previous example, the Feedback Manager detects the error on the previous speculation and updates the last column of the table to B*, since the transition B→B is part of the Usage Plan B*.

4.4 QoS-centric Control Scheme

The third key component of our approach, run by the QoS Module, is the QoS-centric Control Scheme and is responsible for regulating QoS. We first present an overview of the component, describe the procedure, present an example and finally we discuss its internals and pseudocode.

Stavros Papastavrou

4.4.1 Overview

Our approach for regulating QoS is to employ a straightforward QoS-centric procedure. The idea is to initiate at runtime an increase or decrease on the number of fragments that are allowed to materialize per template request so that performance goals are met. In return, this fluctuation on the number of fresh fragments per template affects data quality in terms of QoL and QoSV.

This procedure is essentially symmetric to the QoD-centric OVIS algorithm found in [65, 66], according to which effort is first made to keep data quality within acceptable margins. The reason for adopting such a straightforward QoS-centric approach over a QoD-centric one is because, there is no guarantee that by reducing the overall data quality would yield improved performance. In more detail, a lower acceptable QoD per template does not imply that fewer fragments are materialized, since fragments do not have a uniform data quality value (as discussed in Section 3.3). Instead, by explicitly reducing the number of materialized fragments per template, we secure performance gains since computational resources are immediately spared.

4.4.2 Procedure Description

Two essential factors for the QoS-centric Control Scheme are (a) the maximum tolerable response time and (b) the average response time of currently active user sessions. The former defines a threshold for the latter which, when violated/crossed, triggers the QoS-centric Control Scheme to take corresponding action.

At run time, the QoS-centric Control Scheme periodically checks the average response time of active user sessions. If this is found steadily higher than the maximum tolerable threshold due to increased workload, then the QoS-centric Control Scheme takes action to push it lower by issuing an initial decrease on the maximum number of fragments per template that are materialized until the next check.

Stavros Papastavrou

This decrease is controlled since (a) it is progressively applied to a percentage of active user sessions in each check period and (b) it is comprehensive in the sense that all active user sessions must be affected at least once before issuing a second decrease if necessary. The latter occurs when the initial decrease did not stabilize the average response time below the threshold and an additional decrease is required.

As soon as the average response time is stabilized below the threshold, the decrease is suspended. In this case, the QoS-centric Control Scheme can reverse the procedure by issuing an increase on the maximum number of fragments per template for materialization. Throughout this dissertation, we refer to the practice of applying the decrease to a user as "dropping the user" or "dropping one (or more) fragment" from the user. We refer to "upgrade" for the opposite practice.

4.4.3 Procedure Example

In order to present a simple example of the procedure described above, certain parameters must be set (their semantics are discussed in next). For this example, we define a *QoS Threshold* to be at 200ms and the periodic check on the average response time to occur every W seconds. New user sessions arrive at the system on a linear rate of five per period W. The number of fragments to drop per template is one. The drop is enforced to five user sessions per period W. Finally, we assume that the number of fragments per template and their execution time are approximately the same.

In the example, (illustrated in Figure 34), the initial server workload is 30 active user sessions and the average response time is steadily below the QoS Threshold at about 175ms. As the workload gradually increases toward a peek of 75 user sessions, the average response time exceeds the QoS Threshold at about 55 user sessions. At about 60 user sessions, it stabilizes above the QoS Threshold. This is detected by the QoS-centric Control Scheme which initiates a drop of one

Stavros Papastavrou

fragment per template to five users every W seconds. Consequently, the average response begins to decrease when the drop is enforced to 20 user sessions. However, the drop is still enforced to more user sessions until the average response time is stabilized below the QoS Threshold. This occurs when the drop has been enforced to 45 user sessions. At that time, the QoS-centric Control Scheme begins to work the other way around by upgrading user sessions.

4.4.4 Discussion

4.4.4.1 Procedure Parameters

As seen in the example above, there are a number of parameters that characterize the QoS-centric Control Scheme. More specifically,

- The QoS Threshold. It depends on the application requirements and can be modified accordingly, at any time, by the system administrator. A typical value would be in the order of hundreds of milliseconds [102].

- The interval (W) (or check period W, or drop period W), of the periodic QoS Threshold checks. It depends on the arrival rate of new user sessions which do not follow any known distribution [14, 109]. Bursty or high user arrival rates would require a relatively short interval in order for the system to quickly adjust to the increased workload.

- The percentage of user sessions (UserDiff) per interval W, to which a drop or upgrade is enforced. Similar to interval W, bursty user arrival rates would require a higher value.

- The number of fragments to drop or upgrade per template. As we examine later on, this is handled by the procedure that normalizes the inequalities on the number of fragments per template and fragment execution times.

Stavros Papastavrou

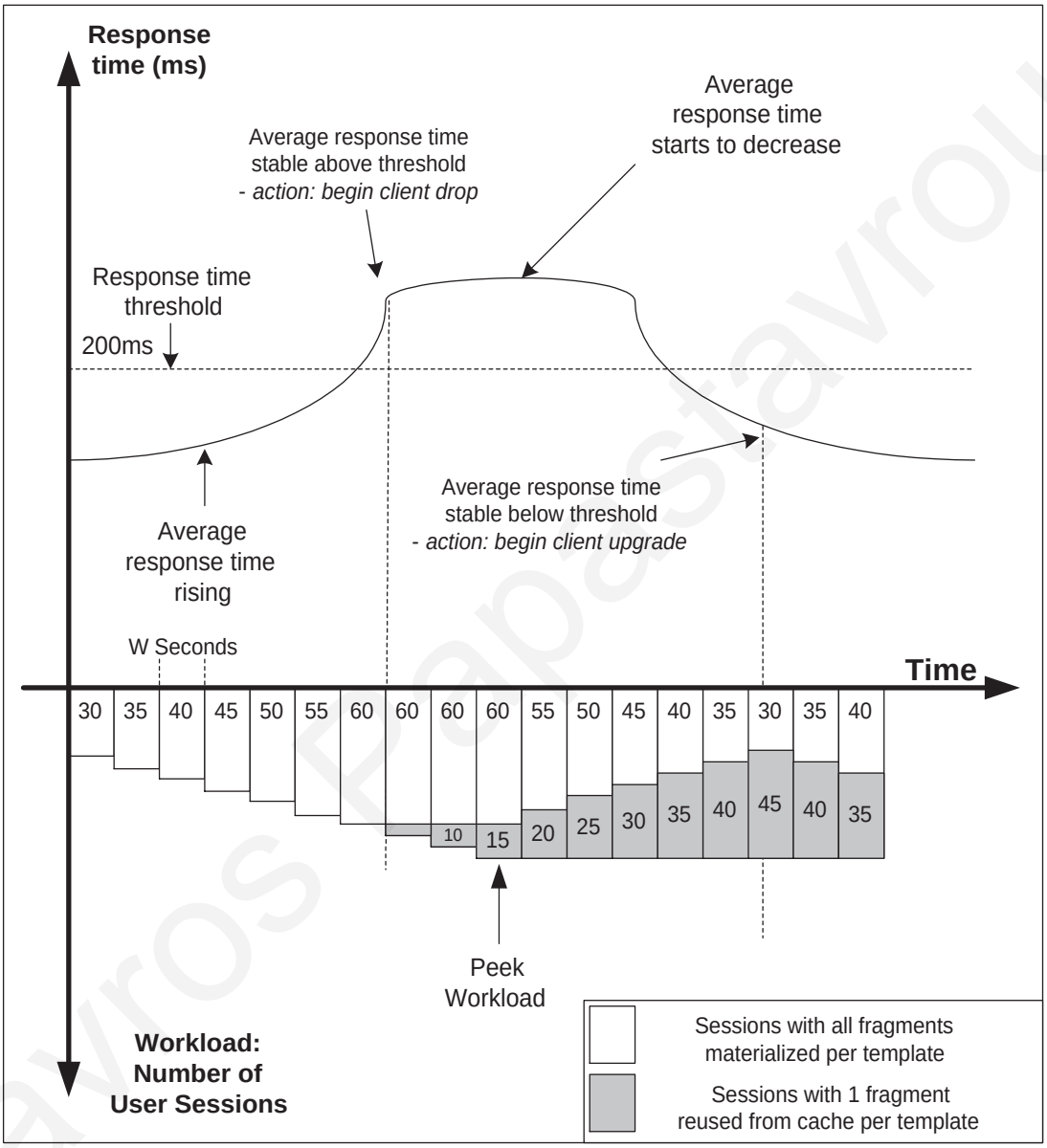

Figure 34: Example run of the QoS-centric Control Scheme for regulating QoS

Stavros Papastavrou

All the parameters above define the "aggressiveness" of the QoS-centric Control Scheme and can be adjusted to respond to different situations and requirements. We elaborate more on this with the following example:

Consider a system which is overloaded with 100 users sessions. A gradual drop of one materialized fragment to all the 100 user sessions has eventually the same effect on the overall QoS with dropping two materialized fragments on the first 50 user sessions. Although this policy may not be fair to the first 50 users, it has two potential benefits/applications: First, it can make the system respond quicker to the sudden workload. Second, it can be applied for profiling purposes where good customers (user who buy more books) must be served at all times with fresh content. In Chapter 6, we present an evaluation our QoS-centric Control Scheme. In the final chapter, we discuss more on potential profiling policies and practices.

4.4.4.2 Normalizing Inequalities

The driving force behind the QoS-centric Control Scheme is the drop or upgrade of users. In our simple example presented in Section 4.4.3, we assumed that the number of fragments per template and their execution time are approximately the same. Under these assumptions, a user is dropped by simply reducing by one the number of materialized fragments per template for that user. In a real-word web database application, however, this is not the case.

Consider a web database application where all its templates have eight fragments each except from `template X` which has 16. Moreover, the execution time of all fragments is approximately the same. When a user, who is already dropped once, requests `template X`, then two fragments of that template should not be materialized.

Hence, when a user who, is already dropped twice, requests `template X`, then four fragments should not be materialized. The same should happen in the scenario where all the templates

Stavros Papastavrou

had exactly eight fragments and the execution time of fragments in `template X` is half compared to that of the fragments in the other templates.

To handle this issue, an abstraction layer that applies a common denominator on how many fragments are dropped per template is required. For this reason, the QoS-centric Control Scheme maintains a **Global QoS Level Index**. Its value reveals the magnitude (or coefficient) of drops per user necessary for the system to maintain QoS and is independent from the number of fragments reused from cache per template. When it is set to zero, it implies that the current trend is to materialize all fragments per template request for all user sessions due to light workload. In other words, the trend is not to have any users dropped. A value of -1 implies that the system is experiencing increased workload an so on.

In addition to the local index, a **Local QoS Level Index** is assigned to every user session to denote how many times a user is dropped. The local indexes of the users are maintained by the materialization algorithms which compute the number of fragments to be reused from cache per template, as we elaborate in the next chapter.

4.4.5 Procedure Pseudocode

Figure 35 displays the pseudocode of the QoS-centric Control Scheme initiated at system startup. Its main task is to maintain the Global QoS Level Index by increasing or decreasing it according to the workload. Depending on the workload, it also sets a system directive necessary to the materialization algorithms.

As system startup, the Global QoS Level Index is set to zero implying that the system is not overloaded. The main loop of the procedure executes every W seconds in which counters that keep track on the number of users that are dropped or upgraded per period W are set to zero (line

Stavros Papastavrou

Running Process **control_Global_QoS_level** ()

```
1   {
2     Global_QoS_level = 0;
3
4     while (True) {
5
6       wait( W seconds );
7
8       // resent counters to ensure that
9       reset_Number_of_Clients_Dropped_or_Upgraded_in_Last_Period();
10
11      if ( overall average response time      SteadilyAbove   threshold )
12      {
13         // this directive is used by the QLS algorithm
14         systemDirective =   "    must favor QoS  ";
15
16         if ( all active clients have local QoS_level == Global_QoS_level )
17         {
18              Global_QoS_level = Global_QoS_level - 1;
19         }
20      }
21      else if ( overall average response time      SteadilyBelow   threshold )
22      {
23          systemDirective =    "    must favor content quality    ";
24
25         if ( all active clients have local QoS_level == Global_QoS_level )
26         {
27             Global_QoS_level = Global_QoS_level + 1;
28          }
29      } // end if
30    } // end while
31  } // end process
```

Figure 35: The QoS Controlling Loop

Stavros Papastavrou

9). This is necessary since the materialization algorithms must not allow for more users to drop or upgrade than the desired percentage (UserDiff).

Then, the current average response time is compared against the QoS Threshold and corresponding action is taken (lines 11). If it is found to be steadily higher than the threshold, then a system directive to favor QoS is issued. Following that, the procedure compares the Global QoS Level Index with the Local QoS Level Index of all user sessions (line 16). If they are equal, then the Global QoS Level Index is further reduced by one. This is the case when all users are already equally dropped and, still, the average response time is higher than the QoS Threshold.

4.5 Chapter Summary

This chapter presented the three key components that characterize our approach. We presented two new data quality metrics of QoL and QoSV that capture the semantics of content dependencies. The former quantifies the ability of a user to request the next template using the links in the fresh fragments of a template. The latter quantifies the set-wise freshness of fragments inside a template.

Usage Plans and the profiling mechanism are the means of speculating on the next template that a user will request. The speculation is a key component of our algorithms, since it is directly related to QoL.

Finally, we presented the QoS-Centric Scheme that regulates QoS by increasing and decreasing the number of fragments that are materialized on every user request. The goal of the QoS-centric scheme is to keep the average response time around a predefined threshold.

Stavros Papastavrou

Chapter 5

Materialization Algorithms

This chapter presents our two proposed materialization algorithms for balancing QoS with QoL and QoSV. The first one provides provision for link dependencies by balancing QoS with QoL. The second provides additional support for set-view dependencies by balancing QoS with both QoL and QoSV. We also elaborate on strategies on how to deal with cached fragments containing outdated links necessary for user navigation.

5.1 QLS: The QoL-sensitive Algorithm

5.1.1 Overview

The QLS algorithm is sensitive to link dependencies meaning that its goal is to facilitate seamless user navigation even at high workloads where more fragments per template are served from cache. Recall that seamless user navigation relates to the ability of a user to request the next template in its request sequence. The QLS algorithm materializes and serves (a) the appropriate quantity and (b) the appropriate mixture of fresh/cached fragments per template request for all users in order to balance QoS with QoL. QLS stands for Quality of Link-Sensitive.

Stavros Papastavrou

The QLS algorithm executes at the Materialization Algorithms Module and relies on two parameters: The first one is the current server workload expressed in terms of a system directive computed by the QoS-centric Control Scheme at the QoS Module. This parameter is necessary for the QLS algorithm to compute the right quantity of fragments to materialize per template request and therefore maintain acceptable QoS levels. Right quantity of fragments refers to the maximum quantity of fragments allowed to materialize per template given the current server workload.

The second parameter is a speculation on the next template that a user will request, facilitated at the Speculation Profiling Module by employing the Usage Plans of the web database application. This parameter is necessary for the QLS algorithm to compute, given the quantity, the appropriate mixture of fragments to materialize per template request. Appropriate mixture refers to the fresh fragments that maximize the QoL of their containing template in respect to a target template. Maximized QoL implies increased possibility for seamless user navigation.

An overview of the QLS algorithm is listed in Figure 36. An instance of the QLS algorithm is instantiated and attached to every new user session. Triggered on every user request for a template, the algorithm's main loop secures first QoS by computing the right quantity of fragments to materialize in the requested template (line 10). To achieve that, it requires knowledge on the current server workload (line 6) which may result on either a drop or an upgrade to the user (line 8).

Following that, the algorithm selects the appropriate mixture of fresh/cached fragments in the requested template that maximize QoL (line 15) toward the speculated template that the user will request next (line 12). Finally, the selected fragments are materialized and the generated content is served to the user.

In the rest of this section, we present how the QLS algorithm selects the right quantity and the right set of fragments to materialize. Then, we present the QLS algorithm in detail.

Stavros Papastavrou

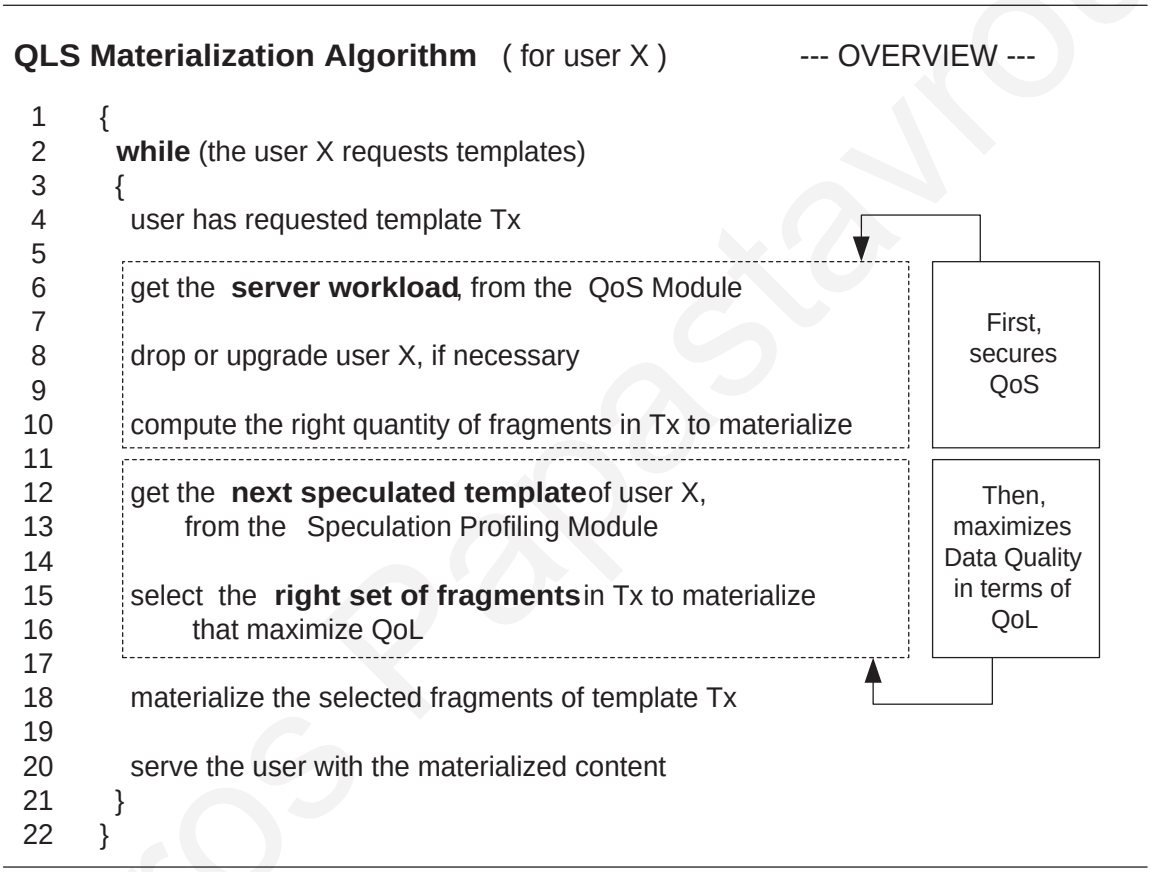

Figure 36: Overview of the QLS Algorithm

Stavros Papastavrou

5.1.2 Securing QoS

The first consideration of the QLS algorithm is to secure QoS by computing the appropriate quantity of fragments per template request that are allowed to materialize. Recall that non-materialized fragments are reused from cache sparing resources for promoting QoS under heavy workload. This procedure relies on a system directive, computed at the QoS Module, that instructs the QLS algorithm to favor either QoS or data quality.

Depending on this system directive, the QLS algorithm may drop or upgrade a user relatively to the Global QoS Level Index. As discussed in Section 4.4.4, the execution time of all fragments as well as the number of fragments per template are uneven. As a consequence, a user cannot be dropped from a Local QoS Level Index zero to -1 by simply reusing one fragment from cache per template request. In addition, even if the number of cached fragments to reuse was known, the QLS algorithm cannot arbitrary select any fragments. This is because, the selected fragments must additionally maximize QoL, as we cover in detail in the next section.

To solve this issue, the QLS algorithm lists all combinations of fresh/cached fragments per template into groups tagged with a QoS level index. The intuition is that, the combinations in each group require approximately the same execution time. The combinations in groups with a lower QoS level index require less time to execute. This grouping allows for the QLS algorithm to significantly narrow down the number of alternative combinations of fresh/cached fragments relating to the Local QoS Level Index of the user.

Figure 37 illustrates an example grouping of the combinations of fresh/cached fragments in a template with four fragments. For ease of presentation, the example assumes that the execution time of fragments is approximately the same (approximately 50ms). Each combination is called a *Materialization Plan* (MP) and consists of a 1s and 0s that relate to the rank of the fragments

Stavros Papastavrou

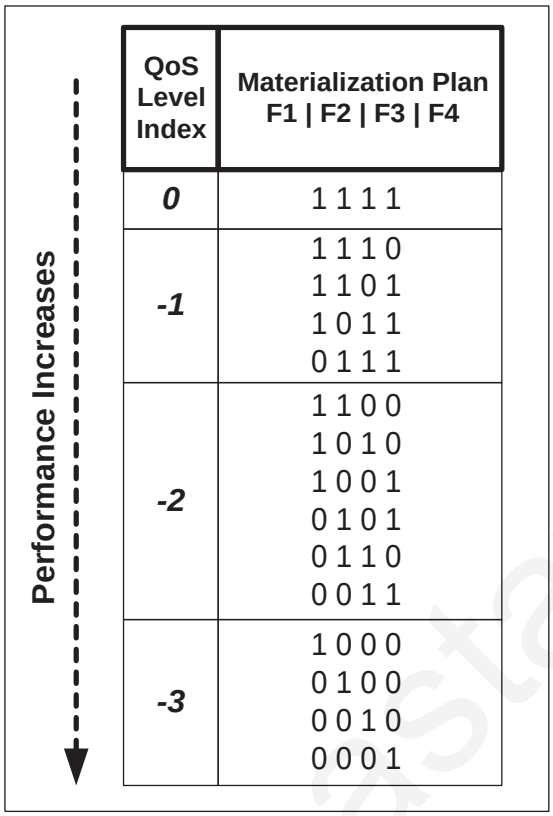

Figure 37: Materialization Plans (Combinations of Fresh/Cached Fragments) Grouped by QoS Level Index.

in the template. The MP '1111' of a template with four fragments implies that all fragments are materialized. The MP '1101' implies that all fragments are materialized except from the second from last that is retrieved from cache. Since each fragment requires approximately 50ms to materialize, the MPs in QoS Level 0 require 200ms to materialize, in QoS Level -1 require 150ms and so on.

Fragments that are generation-dependent on others that are not materialized in a MP, are also not materialized. For example, if the first fragment is not materialized in MP '0111', having the last fragment generation-dependent on it, then the MP becomes '0110'. Consequently, the MP '0111' would be deleted from the candidate MPs of QoS Level Index -1 in the table of Figure 37.

Stavros Papastavrou

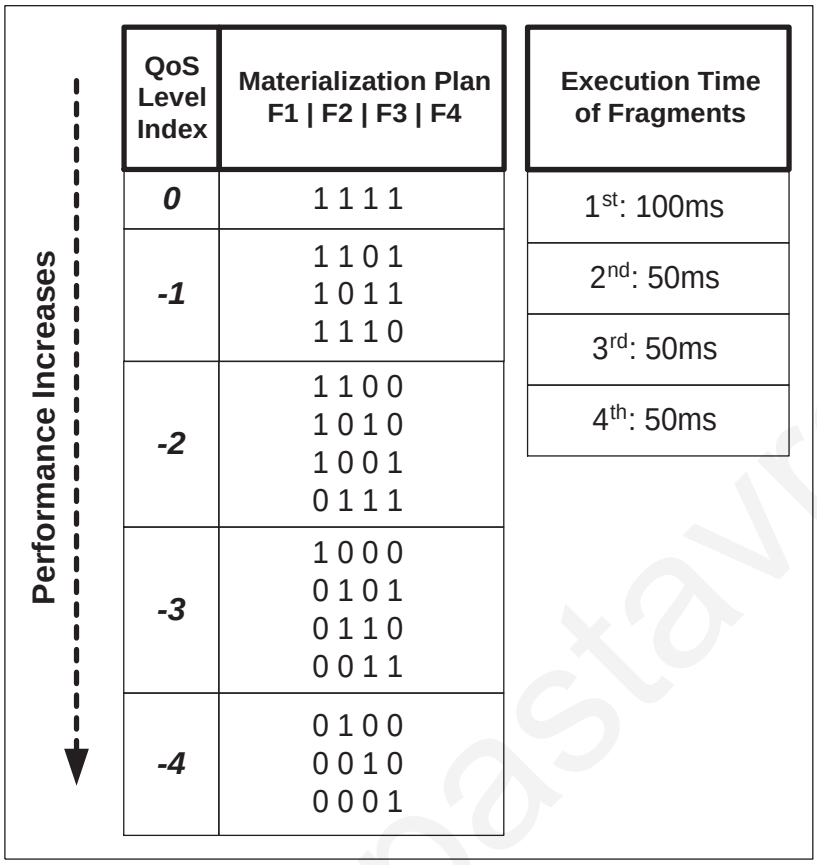

Figure 38: Materialization Plans (Combinations of Fresh/Cached Fragments) Grouped by QoS Level Index. The first fragment requires approximately the double time to materialize compared to each one of the rest fragments.

In the scenario where the execution time of fragment is not approximately the same, then the grouping of the MPs in the table of Figure 37 would differ. Consider the example where the first fragment requires approximately 100ms to materialize as opposed to the rest three fragments that require approximately 50ms. In this case, the grouping of the MPs (as seen in Figure 38) would require five QoS Level Indexes according to the materialize time of the MPs. This is because all the possible combination of fresh/cached fragments produce five groups of MPs with different execution times. Also note that MPs in lower QoS levels do not necessarily have less fresh fragments than other MPs on higher QoS levels.

Stavros Papastavrou

5.1.3 Securing QoL

Having secured QoS by selecting a group of candidate materialization plans, the QLS algorithm must then maximize QoL. This is achieved by selecting the appropriate materialization plan from that group, whose fresh fragments maximize QoL toward the speculated template that the user will request next. Recall that the speculated template is computed by the Speculation Profiling Module.

To achieve that, the QLS algorithm employs a structure called the *MP Selection Structure* which is an extension to the table that groups the materialization plans into QoS Level Indexes of Figure 37. This structure contains additionally the QoL values for each materialization plan toward a target template. Figure 39 illustrates the MP Selection Structure for template `search.dyn` (showing only four fragments) with QoL values computed for target templates `search.dyn` and `viewBook.dyn`.

The QoL values for each MP are computed according to the formula QoL(T_s,T_d) presented in Section 4.2.2. Recall that this formula sums up the QoL importance weights of materialized fragments (with 1s) in template T_s toward a target template T_d. For this example, fragment F1 is link-dependent to `search.dyn` (S) with an importance factor or 0.5 and to `viewBook.dyn` (V) with 0.6. Fragment F2 is 0.3 link-dependent to S and 0.1 to V. Fragment F3 is only dependent to V with 0.3. Finally, F4 is only dependent to S with 0.2. The template `shobBox.dyn` (B) is not present in the MP Selection Table since there are no fragments in `search.dyn` with link dependencies to it.

Having narrowed the candidate materialization plans down to a group, the QLS algorithm maximizes QoL by selecting the MP with the highest QoL value toward the speculated template.

Stavros Papastavrou

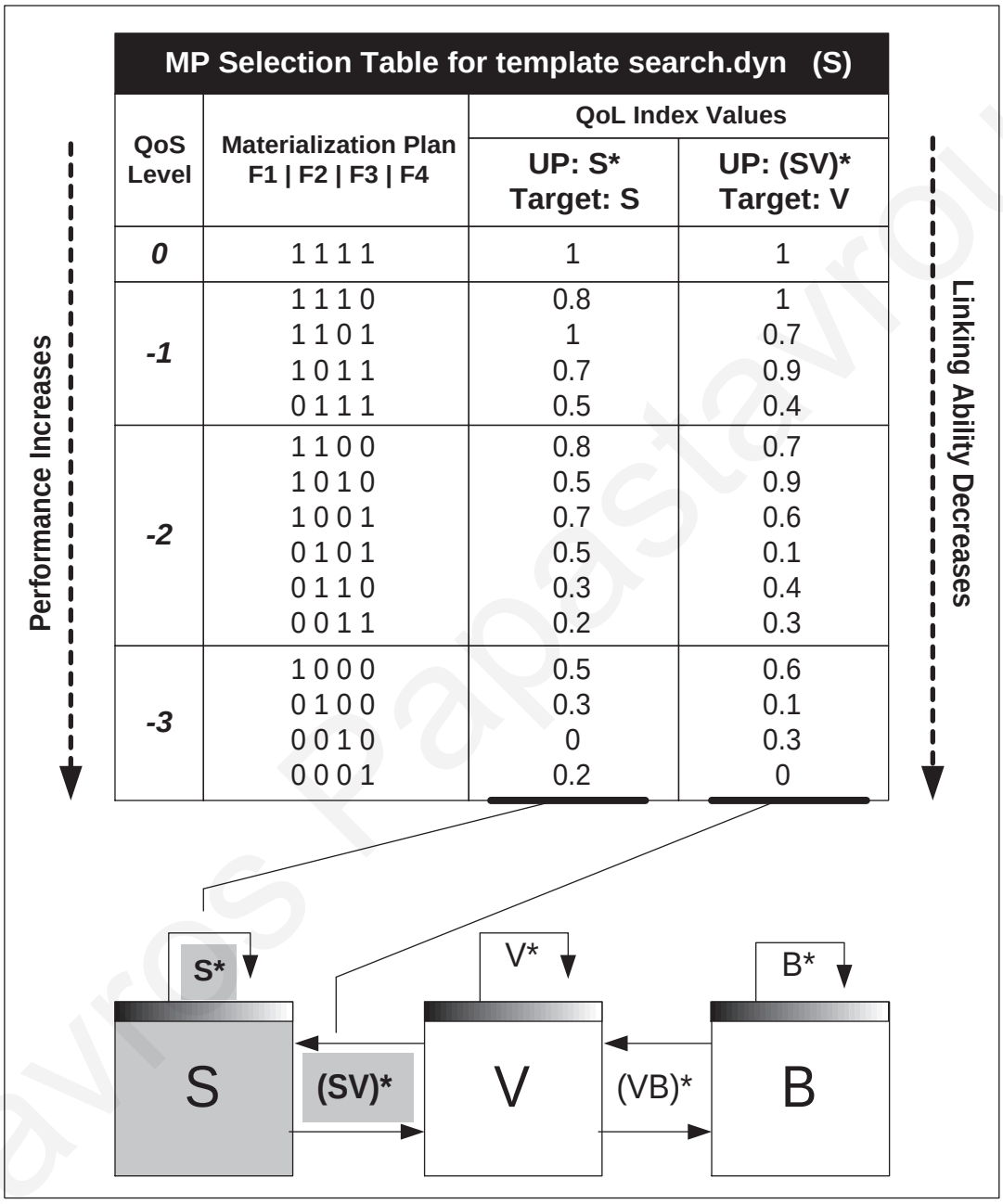

Figure 39: The MP Selection Table for Template `search.dyn`.

Stavros Papastavrou

Finally, this MP denotes which fragments should be materialized and which to be reused form cache.

5.1.4 Pseudocode of the QLS Algorithm

In this section, we present a more thorough version of the QLS algorithm pseudocode in which details, such as how users are initialized are how users dropped or upgraded, are shown (see Figure 40. At start, the user is assumed to be already viewing a default template T_{cur}. In addition, the user's current Usage Plan UP_{cur} is set to a default one. The initial value of the Local QoS Level Index of the client is set to the Global QoS Level Index.

The main loop of the QLS algorithm executes as long as the user request a template, denoted by the T_{tar} variable. First, the QoS directive is acquired by the QoS Module (line 13). If the directive is to favor performance, then the user is dropped if the following two conditions hold: the Local QoS Level Index of the user is higher than the Global QoS Level Index and the percentage of users dropped (UserDiff) in the current check period (W) is not reached (lines 16-18). The analogous procedure is followed for upgrading a user (lines 20-22). Having computed the Local QoS Level Index of the user, the group of MPs is acquired from the MP Selection Table of the requested template T_{tar}, given the Local QoS Level Index of the user (line 26).

Then, the speculated target Usage Plan UP_{tar} of the user is acquired from the Speculation Profiling Module, given the current UP, the current and requested templates of the user (line 29). The speculated UP reveals the next requested template. Following that, the MP that maximizes the QoL is selected toward the speculated template out of the group of candidate MPs (line 32).

Finally, the fragments of the requested template T_{tar} that correspond to the 1s in the selected MP are materialized (line 34). With the rest fragments pulled from cache, the complete dynamic

Stavros Papastavrou

QLS Materialization Algorithm (for user X)

```
1  Global Variable:
2  UserDiff=20%;   // the percentage of users that are allowed to drop per period
3  Global_QoS_Level_Index; // the local QoS Level Index (managed by the QoS Module)
4  {
5    Tcur = getDefaultTemplate();                    // the current template for user X
6    UPcur = getDefaultUP();                         // the current usage plan for user X
7    Local_QoS_level =  getGlobalQoS_level()   // the local QoS level for user X
8
9  // while the user requests a target template Tar
10 while (Ttar) {
11
12   // get QoS server directive, from the QoS Module
13   QoS_Directive =  getQoS_directive();
14
15   // compute next local QoS level for user X
16   if ( QoS_Directive == "must favor QoS" & Global_QoS_level<Local_QoS_level)
17       if ( percentage of users dropped in current period W     < UserDiff of active users)
18           Local_QoS_level-=1;   // drop user X
19  else
20   if ( QoS_Directive == "must favor content quality" & Global_QoS_level>Local_QoS_level)
21       if ( percentage of users upgraded in current period W < UserDiff of active users)
22           Local_QoS_level+=1; // upgrade user X
23
24   // get the group of candidate materialization plans
25   // that match the local QoS Index of user X
26   cancidateMatPlans =   SelectionTables[Ttar]  [Local_QoS_level]
27
28   // speculate a usage plan - using the profiling mechanism
29   UPtar = mineUsagePlan (UPcur, Tcur, Ttar);
30
31   // get materialization plan with maximum QoL for UPtar
32   MP = getMP_with_max_QoL(cancidateMatPlans, UPtar);
33
34   materialize(MP); // materialize template according to the MP
35   // prepare for next request
36   Tcur = Ttar;
37 }}
```

Figure 40: A more Detailed Pseudocode of the QLS Algorithm.

Stavros Papastavrou

web page is served to the user. To prepare for the next request, the target template T_{tar} substitutes T_{cur}.

5.1.5 Discussion

Initially, the QLS algorithm assumes that a new user is already viewing a default template, which is considered to be the entry point of a web database application. In our bookstore application, the entry point is the `search.dyn` template. In addition, the new user would be on default Usage Plan. In our case, this would be the S*, since we expect the user to perform a few searches before deciding to view any resulting book. Both default values are necessary for the QLS algorithm so that a proper speculation is made. They can be set offline by the system administrator according to the application semantics.

The Local QoS Level Index of a new user is set to the Global QoS Level Index. In other words, a new user is automatically dropped as many times as necessary to immediately comply with the current QoS directives, as this is the case with currently active users. Moreover, the algorithm does not allow for more than a parameterized percentage of users (UserDiff) to drop per period W. This parameter depends on the arrival rate of users and can be dynamically adjusted so that the system can adapt to sudden increased workload. We review this parameter in the next chapter.

5.2 The QoSV variation of QLS

5.2.1 Overview

The QoSV-enhanced variation of the QLS materialization algorithm balances QoS and content quality in terms of both QoL and QoSV. Similar to QLS, its goal is to enable seamless navigation for users at high workloads with the additional goal of satisfying set-view dependencies where

Stavros Papastavrou

possible. We add to the QLS algorithm the extra goal of selecting the materialization plan with the highest possible index for QoSV with respect to a relax factor on QoL.

To facilitate QoSV, an extra column is added to the MP Selection Structure (last column in Figure 41) to record the QoSV index value of every MP. This value is computed according to the formula QoSV(T) in Section 4.2.3 that sums up the QoSV importance weights of synchronized fragments that are set-view dependent in template T. In our example, the 1st fragment has a set-view dependency weight of 0.6 to the 3rd fragment and 0.4 to the 4th fragment.

A relax factor of 0% implies that the algorithm considers only the MPs with the highest possible QoL index. For example, if the user's Local QoS Level index is -1 and the speculated Usage Plan is (SV)*, then only the MP '1110' with QoL equal to 1 and QoSV equal to 0.6 is considered. However, a relax factor of 10% implies that the algorithm **additionally** considers the MP '1011', which has QoL equal to 0.9 and QoSV equal to 1. In this way, the relax factor reduces the linking ability of candidate MPs in order to improve QoSV. In the example, a 10% relax on QoL yields a 40% gain on the QoSV of the selected MP.

The only variation in the pseudocode of the QLS algorithm is the substitution of the function

```
getMP_with_max_QoL(...)
```

at line 32 with

```
getMP_with_max_QoL(..., relaxFactor)
```

which returns the MP with the maximum QoSV index and the maximum, minus the relaxFactor, QoL value.

Stavros Papastavrou

MP Selection Table for template search.dyn (S)				
QoS Level	Materialization Plan F1 | F2 | F3 | F4	QoL Values		QoSV Values
		UP: S* Target: S	UP: (SV)* Target: V	
0	1 1 1 1	1	1	1
-1	1 1 1 0	0.8	→ **1**	**0.6**
	1 1 0 1	1	0.7	0.4
	1 0 1 1	0.7	→ **0.9**	**1**
	0 1 1 1	0.5	0.4	0
-2	1 1 0 0	0.8	0.7	0
	1 0 1 0	0.5	0.9	0.6
	1 0 0 1	0.7	0.6	0.4
	0 1 0 1	0.5	0.1	0
	0 1 1 0	0.3	0.4	0
	0 0 1 1	0.2	0.3	0
-3	1 0 0 0	0.5	0.6	0
	0 1 0 0	0.3	0.1	0
	0 0 1 0	0	0.3	0
	0 0 0 1	0.2	0	0

Figure 41: Selection Table of Materialization Plans for Template `search.dyn` with both QoL and QoSV Indexes. For a QoS level=-1, Usage Plan (SV)* and a QoL relax factor=10%, the plan '1011' is selected instead of '1110'.

Stavros Papastavrou

5.2.2 Discussion

The driving force behind the QoSV variation of the QLS algorithm is the relax factor on QoL. A lower value for the relax factor implies that the algorithm selects fragments for materialization based primarily on their link dependencies and secondary on their set-view dependencies. As a result, it serves pages with fewer broken link dependencies and more unsynchronized set-view dependent fragments.

On the other hand, a higher value for the relax factor implies that the algorithm selects fragments for materialization based primarily on their set-view dependencies and secondary on their link dependencies. As a result, it serves pages with more broken link dependencies and less unsynchronized set-view dependent fragments.

In other words, the former configuration better suites applications that require frequent navigation between templates. On the other hand, the latter configuration better suites applications whose requirements on consistent content are more important than frequent navigation between templates. In the next chapter, we present our experimental findings on the QoSV variation of QLS and elaborate on how the QoSV variation applies to various web database applications.

5.3 Outdated Links in Cached Fragments

In this section, we elaborate on the problems caused by the unavoidable transmission of cached content to the users due to the balancing of QoS with QoL and QoSV. Subsequently, we discuss methods for solving those problems.

Stavros Papastavrou

5.3.1 Transmission of Cached Fragments

Recall that at low workload, the Global QoS Level Index and the Local QoS Level Index of users are equal to zero. This implies that all fragments in every requested template are materialized. As a result, all the links in a template toward any other possible template are served fresh and therefore, a user can navigate irrespectively of the speculation on its Usage Plan by the Speculation Profiling Module.

However, as workload increases, the Global QoS Level Index begins to drop. As a result, users begin to receive a mixture of freshly materialized and cached fragments in their requested templates. According to the QLS algorithm, the first fragments to be reused from cache are those that do not contain any links to the next speculated template. As workload further increases, the QLS algorithm begins to reuse from cache even fragments containing necessary links to the next speculated template.

Clearly, cached fragments contain outdated (or invalid, or broken) links that cause navigation problems to users. Two examples of outdated links from our motivating application are: (a) a URL link for adding an irrelevant book in the shopping box of a user and (b) a FORM for repeating a book search with irrelevant parameters.

A user receives an outdated link, as part of a cached fragment, in the following two cases:

- In the first case, the speculation on a user's next requested template was wrong and the required link is part of a fragment with no link dependencies to the falsely speculated template. As discussed earlier, this case occurs mostly on lower workloads, when the first fragments to be reused from cache are those containing links to the speculated template.

- In the second case, the speculation was correct, however, heavy workload forced the algorithm to reuse from cache even fragments with links to the correctly speculated template.

Stavros Papastavrou

5.3.2 Handling of Outdated Links

A required link that is served outdated, halts (or stalls) the user's session in the sense that if it was followed it would lead to unwanted or inconsistent results. The solution to this problem is twofold:

- First, a policy is required concerning the method through which the cached fragment containing the outdated link is rendered on the user's browser. This is necessary since the user must be made aware of the fragments that are not fresh.
- Secondly, a mechanism is required to support explicit materialization of the cached fragment on the server so that the contained required link is fetched and rendered fresh on the user's browser.

5.3.2.1 Explicit Fragment Materialization

We begin with discussing the second issue of explicit fragment materialization. Our approach on this issue is to provide support on the server side by allowing for a special HTTP GET request with special parameters to target *only* a particular fragment. In addition, a requirement is that the user must be currently viewing its containing template.

This requirement is essential since the explicit materialization of a fragment may require prior results stored in session variables. These results were computed during the materialization of other fragments of the containing template. In order to avoid data dependency problems, generation dependencies must be taken into account.

Stavros Papastavrou

5.3.2.2 Rendering Cached Fragments

An appropriate rendering policy is required since the end-user must be made aware that a content fragment on the browser is not fresh. In addition, an interaction policy is required so that a user is provided with a related mechanism to materialize a required fragment. We propose a number of policies, illustrated in Figure 42:

1. A cached fragment can be rendered on the user's browser in a read-only fashion and surrounded, e.g., by dotted lines so that the user cannot immediately follow an invalid link. A first click on the fragment triggers the special HTTP GET request that fetches a fresh version of the fragment containing the valid link.

2. The cached fragments are rearranged at the end of the page (related support by the principle of polymorphism, Chapter 2). Again, a first click on the fragment triggers the special HTTP GET request to fetches the fragment fresh.

3. A quite different policy is to render on the user's browser only the freshly materialized fragments. Links with related descriptions can substitute the missing fragments. The links trigger the explicit materialization of the fragments.

The modification of a client browser to support a rendering policy is not in the scope of the present dissertation. However, since the implementation of a policy is necessary for as to perform our experimental evaluation, our browser emulators provide support for issuing the special HTTP GET request that explicitly materializes and fetches a fragment. This issue, along with the server-side support for the special HTTP GET request, is elaborated in Chapter 6.

Stavros Papastavrou

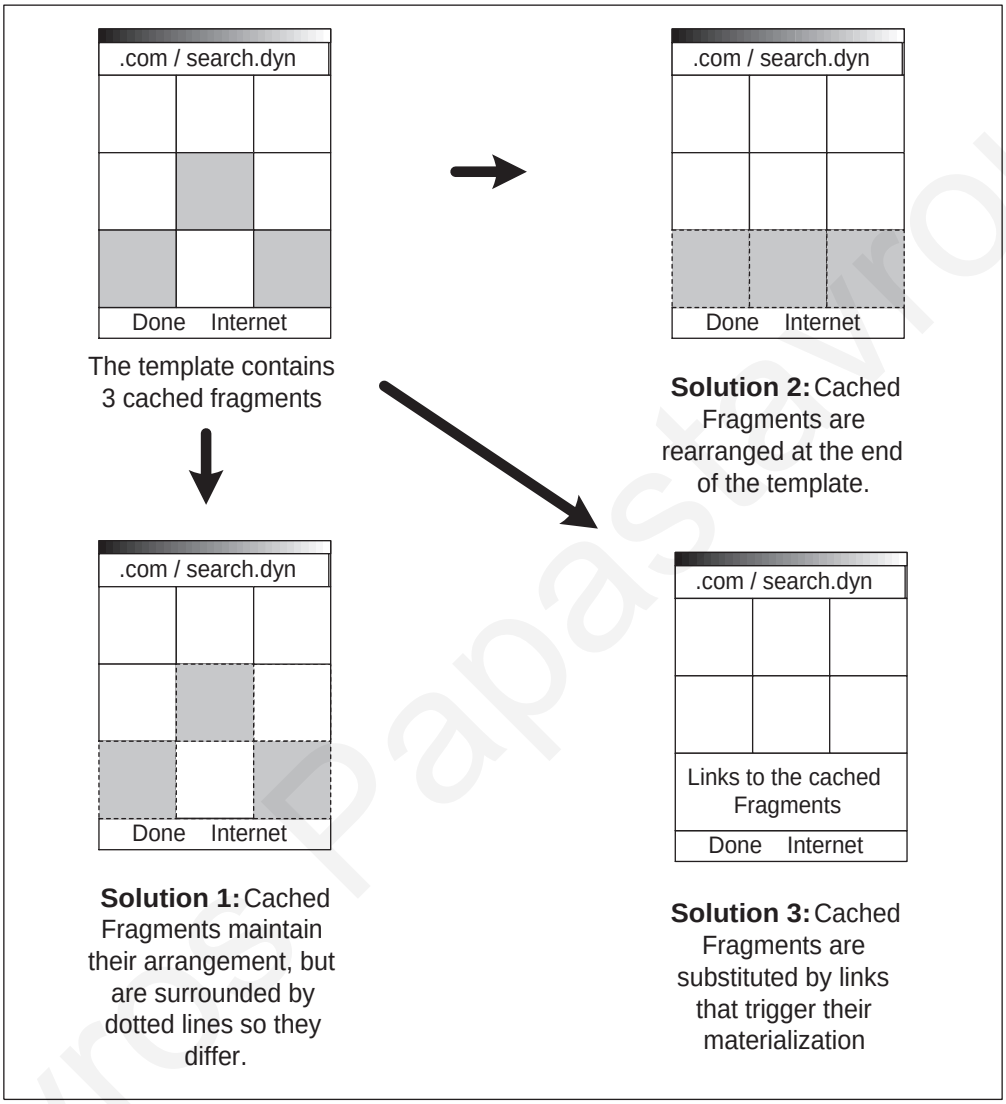

Figure 42: Rendering Policies for Cached Fragments (Grey Squares) Containing Invaliding Links: (1) Cached fragments maintain their position in the page and are surrounded by dotted lines so that they differ. (2) Cached fragments are surrounded by dotted lines so that they differ and are rearranged at the end of the page. (3) Cached fragments are substituted with descriptions and corresponding links that trigger their explicit materialization. Some policies can be combined.

Stavros Papastavrou

5.4 Chapter Summary

Our proposed QLS algorithm and its QoSV variation were presented in this chapter. Both algorithms use a selection structure for identifying the appropriate set of fragments per template for materialization. This set of fragments promotes QoL for the QLS algorithm and additionally, QoSV for its variation. The algorithms compute the maximum possible number of fragments that can be materialized per template in cooperation with the QoS-centric controlling loop presented in Chapter 4.

Both our balancing algorithms inevitably cause the transmission of cached fragments to users some of which contain outdated links required for user navigation. To this end, propose server-side and user-side support for a special HTTP GET request that explicitly materializes an arbitrary fragment form any template. In the next chapter, we measure the performance of our algorithms on various metrics.

Stavros Papastavrou

Chapter 6

Experimental Testbed and Evaluation

In this chapter, we describe the experimental platform that we built in order to evaluate our algorithms and present the findings of our evaluation. Specifically, we first present the server-side platform, the experimental web content, the user-side model and the synthetic workload. Subsequently, we perform an evaluation of the QoS-centric Control Scheme. Then we present our experimental findings by comparing our QLS and its QoSV variation algorithms to the traditional QoD approach. Finally, we discuss the circumstances under which our materialization algorithms are applicable to various web database applications.

6.1 Experimental Testbed

6.1.1 Overview

Our experimental platform is not a simulation but rather a real-world emulation of our motivating web database application. The major difference from a real-world bookstore application is that we do not render the HTML content for commercial use on the user browser. The benefits of using an emulation over a simulation to evaluate our approach is that it provides us with more

Stavros Papastavrou

realistic results since side-effects, such as unexpected server behavior and unaccounted parameters, are factored in. It also gives as a clear picture on the requirements and challenges that an e-commerce system of this magnitude requires.

6.1.2 Server-side System Setup

On our main server machine (a dual CPU, 2GB RAM, RAID 0) we developed and deployed a Java-based web server according to the multi-threaded system model: A web server is a high priority process that accepts incoming socket connections from web users. It performs FIFO admission control by delegating the requests to the first available thread out of a pool of worker threads. When all worker threads are busy, excess pending user requests are put into a ready queue for upcoming execution.

The worker threads are implemented at the application server layer and are responsible for template processing as well as running the materialization algorithms. In our implementation, we use a group of ten worker threads (default is between 5 and 15 in commercial systems). Our decision is based on sensitivity analysis performed on the number of threads by measuring the server throughput, the average response time at the server and the user sides and the length of the ready queue. Our analysis showed that fewer than ten threads led to increased response times that are attributed to the increased ready queue size. In other words, the system was under-utilized. In addition, more that ten threads also led to increased response times attributed to system overloading.

Our approach employs an array of active objects which facilitate our algorithms and are implemented at the application server layer. Figure 43 contains an overview of the implementation. In brief, the main active objects are:

- The Statistical and Plan Managers comprise the QoS Module. They are responsible for performance monitoring and execution of the QoS-centric controlling loop.

Stavros Papastavrou

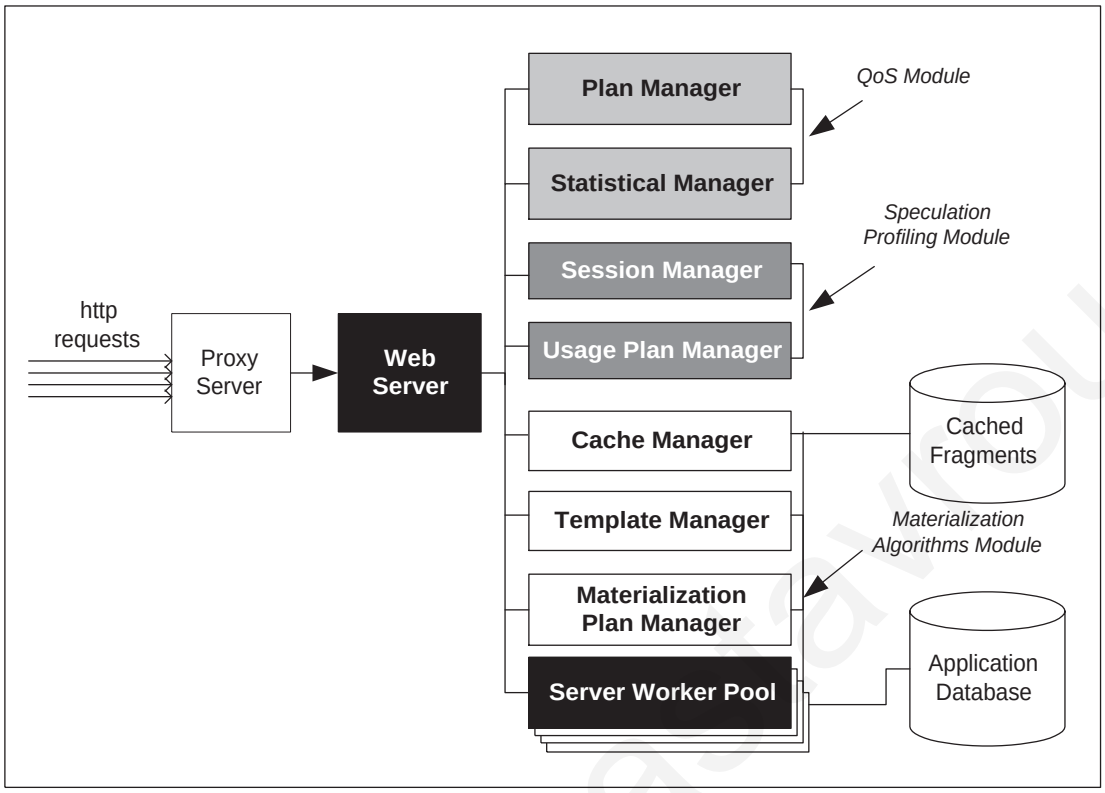

Figure 43: An Overview of the Implementation.

- The Session and the Usage Plan managers comprise the Speculation Profiling Module. They perform user profiling and usage plan speculation.

- The Cache Manager is part of the Materialization Algorithms Module. It handles fragment caching and serves content reuse requests.

- The Template Manager is also part of the Materialization Algorithms Module. It is responsible for template parsing and extraction of dependencies which are sent to the Materialization Plan Manager to construct the MP Selection Tables.

As stated in Section 5.3.2, our server provides support for handling special HTTP GET requests for explicitly materializing any fragment from a template. The Session Manager supports

Stavros Papastavrou

this functionality by providing access to a user's session variables required for fragment materialization. Our experimental web server along with all the active objects is coded using 5000 lines of highly concurrent Java code.

6.1.3 Database Setup

The application database runs on a separate machine (also a dual CPU, 2GB RAM, RAID 0) on the same local network and it is implemented on Microsoft SQL Server 2005. The database holds the data for a Bookstore with more than a hundred thousand books, in addition to data for book availability, authors, shopping baskets, orders etc.

In order to create a realistic book catalog, we manipulated the entries of an English dictionary containing 80,000 words so that they resembled book titles. Authors and other related information were created by manipulating a list of 10,000 available names and by randomly delegating authors to book titles.

6.1.4 Template Setup

We prepared a mixture of templates, each containing eight to ten fragments. The fragments and their content dependencies are setup according to the Bookstore application. Every fragment contains script code that manipulates the results of one read-only query on the application database. In addition, one fragment of the `shopBox.dyn` template executes one update on the application database for placing (or removing) a book in a user's shopping box.

The typical structure of a template file is shown in Figure 44. A static HTML table defines the arrangement of fragments that are placed inside table cells. The existence of a fragment is noted with the new tag `<fragment>` and the parameterized identifier "ID" inside the tag. The identifier refers to a script code block stored separately from the template. The new tags `<link-dep>`,

Stavros Papastavrou

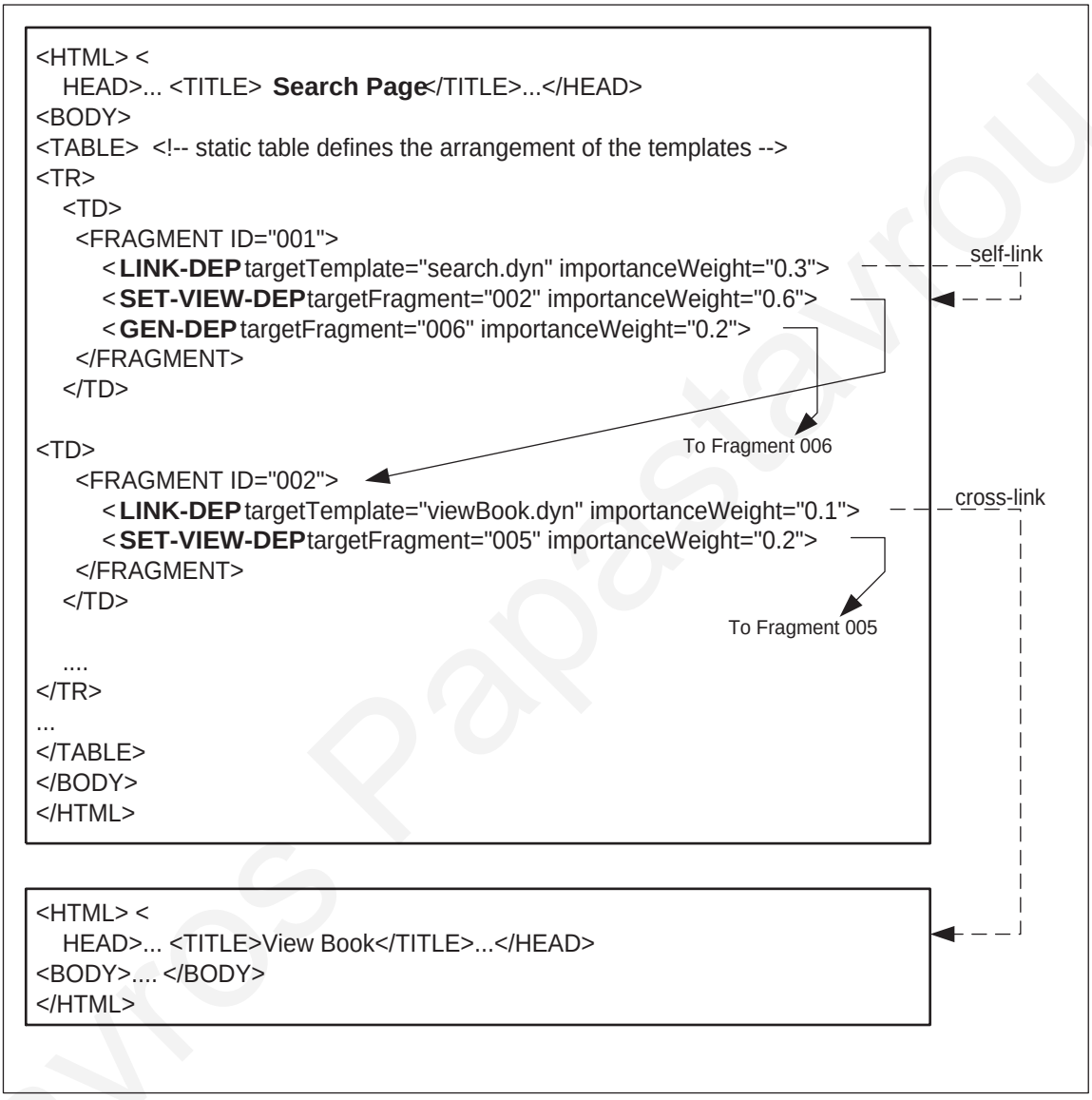

Figure 44: Example Source Code for Template `search.dyn`. Solid arrows show generation dependencies. Dotted arrows show set-view dependencies. The dashed arrow shows the link dependency to template `viewBook.dyn`.

Stavros Papastavrou

<set-view-dep> and <gen-dep> identify the content dependencies of their containing fragment. The targets and the importance weight values of dependencies are identified using tag parameters. The arrows in Figure 44 illustrate the dependencies of the search.dyn template.

We chose this tagged-based approach for marking fragments and their dependencies, which is similar to other popular fine-grained approaches such as ESI. Our fragments are completely decoupled from their containing template and can be incorporated into ESI branching commands.

6.1.5 Web Server Initialization Tasks

At server startup, the Template Manager component of our web server parses and analyzes the structure of the available templates including information on their containing fragments and their dependencies. This information, along with the predefined listing of usage plans of the application, is processed by the Materialization Plan Manager component to construct one MP Selection Table per template. For simplicity and without loss of generality, the execution time of all the fragments in all templates is approximately the same.

In order to facilitate faster selection of materialization plans, the Materialization Plan Manager component uses indexing to group materialization plans by QoS Level and further sort them by QoL and QoSV values. This is necessary, considering that a template with ten fragments has a total of 1023 materialization plans, of which 252 correspond to -5 QoL level. The MP selection tables are stored in main memory and are accessible by other components using a fixed API. The total RAM used by the web server at runtime is measured at approximately about 55MB when server workload is 250 concurrent user sessions.

Stavros Papastavrou

6.1.6 User-side Setup and Synthetic Workload

On a separate machine, we developed and deployed a multi-threaded User Generator engine capable of emulating a large number of user browsers. We chose to create our own user generator engine in order to have greater control over our experiments in terms of user statistical traces and fragment handling. For example, our browser emulators can issue the special HTTP GET request for receiving a fresh version of a cached fragment (as discussed in Section 5.3). Our synthetic workload follows basic principles according to the transactional web e-Commerce benchmark (TPC-W) [78], as discussed in Section 3.4. In particular:

- Popularity of documents follows a zipf-like distribution.
- A small set of documents (around four) account for at least 95% of total user requests.
- This set is stable over time.
- Consecutive user requests occur about every ten seconds.
- The arrival rates of new users do not follow a known distribution.

Our User Generator can spawn on-demand and control an arbitrary number of user browsers that emulate the behavior of web users. The User Generator corresponds to the Remote Browser Emulator (RBE) and the user browsers to the Emulated Browsers (EB) of TPC-W. In our experiments, a user submits a request approximately every 10 seconds [76], analogous to Think Time in TPC-W. The most popular template in our application is `viewBook.dyn` with 50% popularity, followed by `shopBox.dyn` with 25%, `search.dyn` with 12.5% and `used.dyn` with 6.25%.

The order in which templates are requested by a user is correlated to their popularity. We have prepared a number of user request sequences that satisfy the expected zipf-based access frequency of templates. Those patterns are randomly delegated to the emulated users. For better variety

Stavros Papastavrou

on the workload, random users switch between the request sequences. To emulate unexpected user behavior, extra request sequences are periodically delegated to users that do not follow the zipf-based access frequency of templates.

Finally, according to [14, 109], user arrival rates do not follow a known distribution. Instead, they tend to be bursty over certain periods of the day. As we discuss in the next section, we test our QoS Module under different situations.

6.2 Evaluation of the QoS-centric Control Scheme

6.2.1 Overview

In our first group of experiments, we evaluate the QoS-centric Control Scheme, run by the QoS Module, responsible for regulating the QoS of users in terms of server-perceived response time. These tests are needed to prove that our QoS Module is capable of regulating QoS as well as to identify the role of each parameter of the QoS-centric Control Scheme.

In our tests, we use a wide range of server workload and experiment with two different user arrival rates. As detailed in Section 4.4.3, there are a number of parameters that characterize the QoS-centric Control Scheme. These are:

- The QoS Threshold. Expressed in milliseconds, it defines an upper limit for the average response time of serving users.
- The check period W. Expressed in seconds, it defines the frequency at which the average response time is checked.
- The UserDiff. Expressed in percentage of users, it defines the ratio of active users who are dropped per period W when the average response time is found steadily above the QoS Threshold.

Stavros Papastavrou

- The number of fragments to drop per request. As explained in the previous chapter, this is transparently handled by the materialization algorithms and the QoS-centric Control Scheme by the employment of the Global QoS Level Index and the grouping of materialization plans.

A typical value for the QoS Threshold would be in the order of hundreds of milliseconds, depending on the requirements set by the system administrator. For our experiments, we use a 200ms QoS Threshold based on the fact that the average response time for serving a user request in low workload is about 175ms. In this way, we are able to test our QoS Module for a wide spectrum of concurrent user sessions.

The check period W and the UserDiff define the aggressiveness of the QoS-centric Control Scheme in the sense that their values denote how slow or fast the system responds to sudden workload. Given the fact that within approximately ten seconds all active users submit a request, the value of W should also be around ten seconds. This is because successive checks on QoS allow for "enough" time for the user drops to be effective. Furthermore, the value for UserDiff should be in the order of ten for the drop to be effective. In our tests, we experiment with different values for both parameters in order to examine the system responsiveness to linear as well as bursty user arrival rates.

6.2.2 QoS Sensitivity to UserDiff - Linear

Our first experiment tests how the system performs under different UserDiff values using a linear arrival rate for new users (one new user session every 5 seconds). We run the same experiment three times with UserDiff values equal to 20%, 60% and 100%, respectively. For this experiment, the period W is fixed at ten seconds.

Stavros Papastavrou

Parameter	Value
UserDiff	20%, 60%, 100%
QoS Check Period W	10secs
Workload Range	25 to 250 Users
Workload Increment	Linear
User Arrival Rate	1 per 5sec
QoS Threshold	200ms

Table 3: Parameters for the Experiment on QoS Sensitivity to UserDiff - Linear

The routine of the experiment is as follows: Each experiment starts with 25 active user sessions. At this point, the average user request is served within approximately 175ms which is below the QoS Threshold (200ms). In addition, at start the Global QoS Level Index is set to zero meaning that all fragments for all template requests are materialized. We stop the experiment when the number of users reaches the 250 mark. At that point, the Global QoS Level Index is at -5 and the majority of user sessions have a Local QoS Level Index equal to -5. This is because, the constant increment in the number of concurrent users sessions gradually overloads the system which adapts by progressively dropping the QoS Level Index from 0 to -5. Table 6.2.2 summarizes the parameters of the experiment.

The Graph in Figure 45 plots the trend lines of the average response time against the number of active user sessions for all three runs of the experiment. The results show that all UserDiff values are effective in keeping the average response time below the QoS Threshold. However, the system is more aggressive and responsive with a higher UserDiff value in the sense that the average response time is pushed well below the QoS Threshold. In the contrary, a lower UserDiff value keeps the average response time closer to the QoS Threshold.

To interpret these results, we illustrate the distribution of user sessions into QoS levels using stacked area charts. The Graphs in Figures 46 and 47 show this distribution for the UserDiff equal to 20% and 100%, respectively. The former Graph (UserDiff=20%) shows a gradual drop of user

Stavros Papastavrou

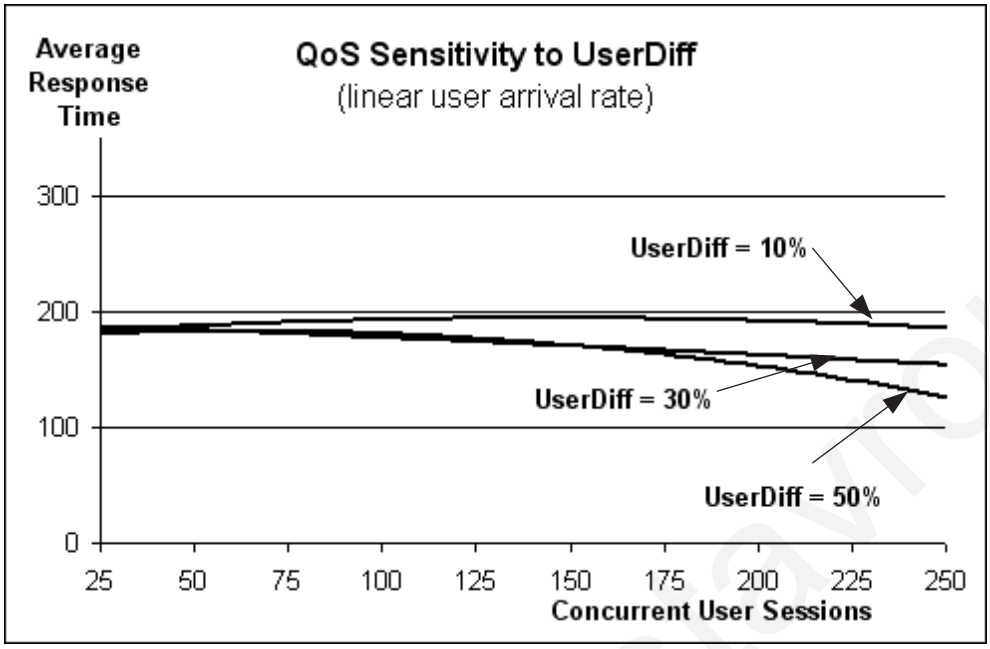

Figure 45: QoS Sensitivity to UserDiff using a Linear User Arrival Rate

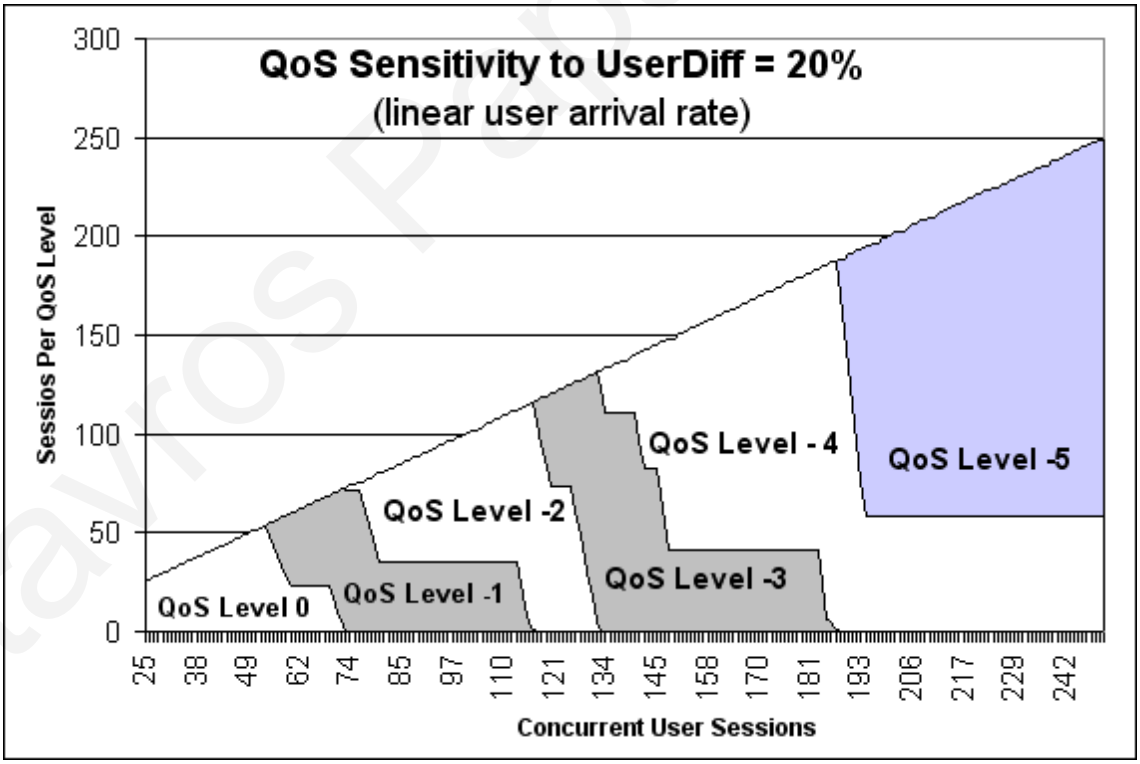

Figure 46: Distribution of User Sessions into QoS Levels - UserDiff = 20%

Stavros Papastavrou

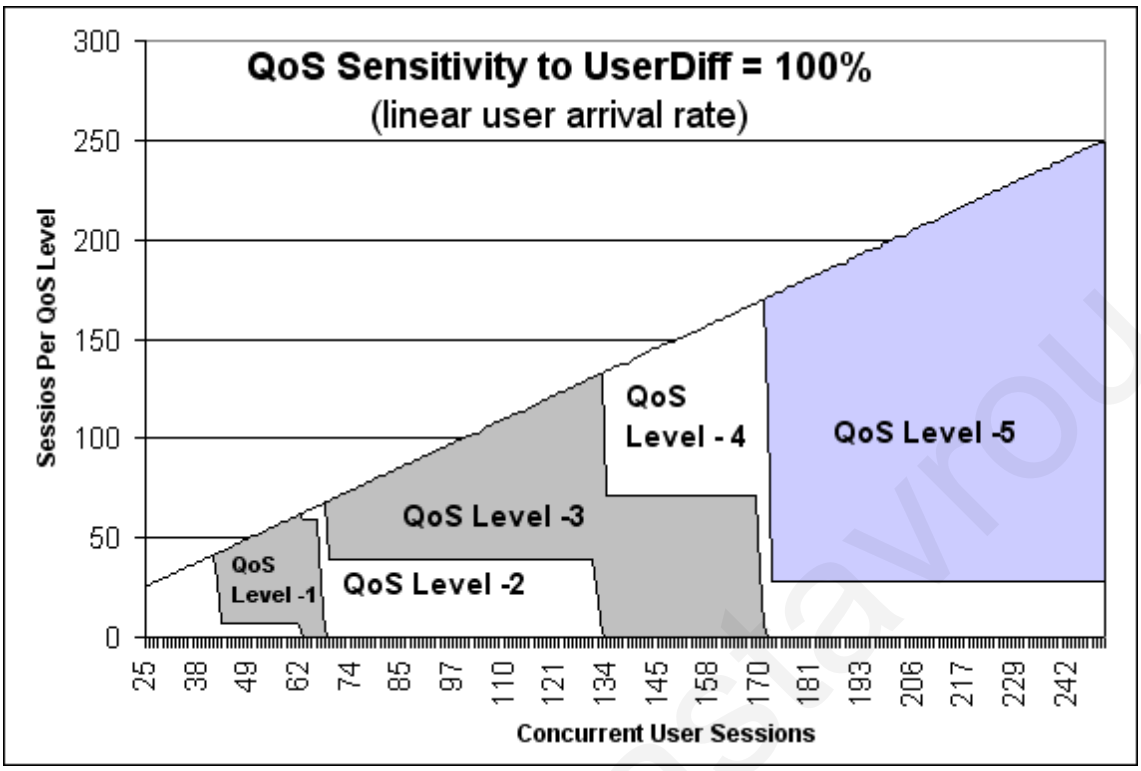

Figure 47: Distribution of User Sessions into QoS Levels - UserDiff = 100%

sessions into lower QoS levels as opposed to the latter Graph (UserDiff=100%) which shows a more rapid drop. As a result, this rapid drop is by far more effective and aggressive in pushing the response time below the QoS Threshold since five times more users are dropped in every period W.

6.2.3 QoS Sensitivity to Check Period W - Linear

In our second experiment, we evaluate the QoS-centric Control Scheme using different values for the length of the check period W and by keeping the UserDiff value fixed. The routine of the experiment is same as previously (parameters summarized in Table 6.2.3). We run the experiment three times using a period W of 5, 10 and 15 seconds, respectively.

The chart in Figure 48 plots the trend lines of the average response time against the number of active user sessions for all three runs of the experiment. The results suggest that the use of

Stavros Papastavrou

Parameter	Value
QoS Check Period W	5secs, 10secs, 15secs
UserDiff	20%
Workload Range	25 to 250 Users
Workload Increment	Linear
User Increment Rate	1 per 5sec
QoS Threshold	200ms

Table 4: Parameters for the Experiment on QoS Sensitivity to Drop Period W - Linear

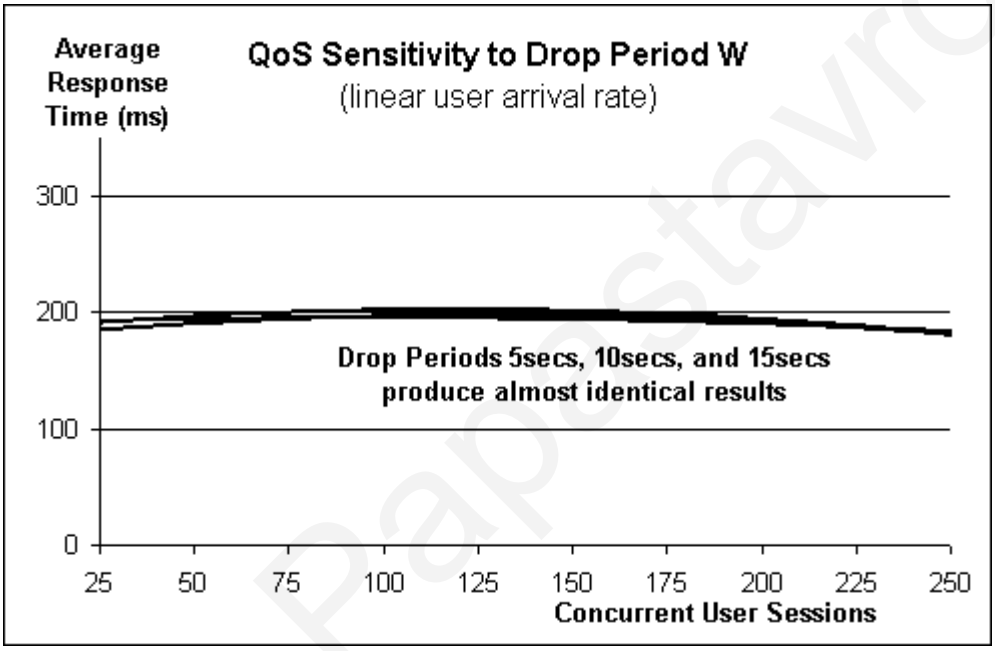

Figure 48: QoS Sensitivity to Drop Period W using a Linear User Arrival Rate

different period W values does not alter the effectiveness of the QoS-centric Control Scheme. To better understand the results, we again illustrate the distribution of user sessions into QoS levels using stacked area charts.

The graphs in Figures 49 and 50 show this distribution for the period W equal to 5 and 15 seconds, respectively. The charts suggest that a smaller check period W has shorter and more frequent drops, as opposed to a larger period W. However, the overall results for each run do not have great difference.

Stavros Papastavrou

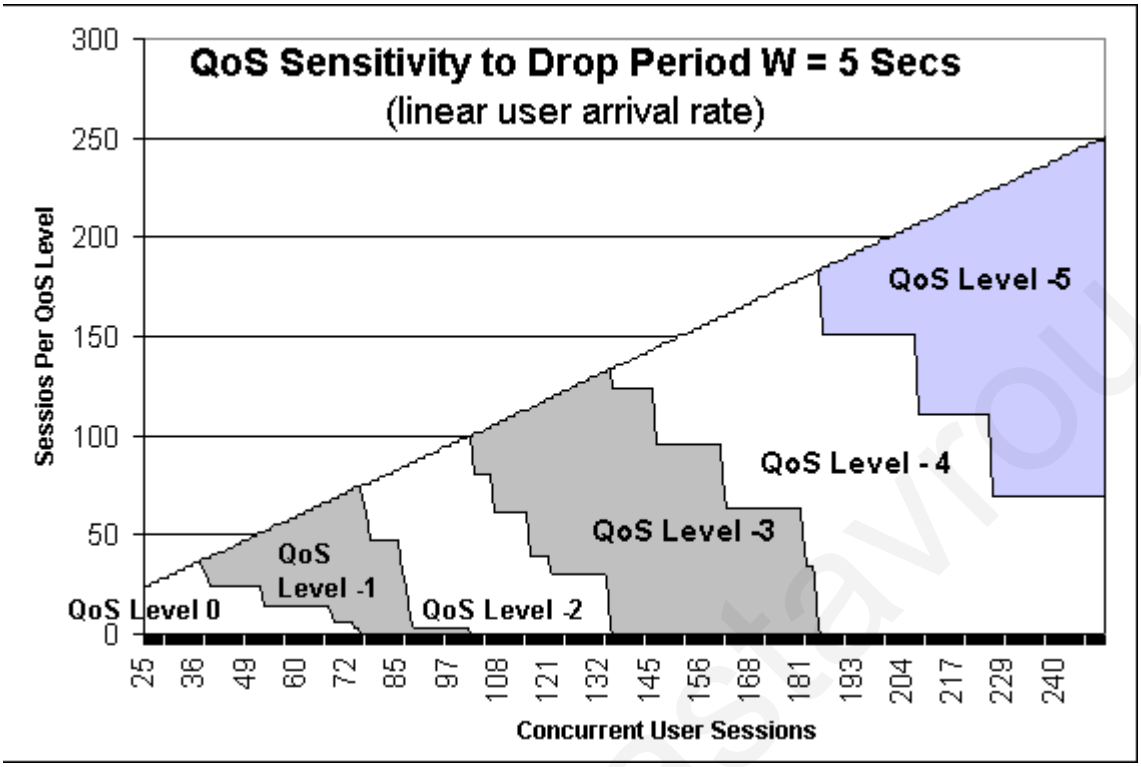

Figure 49: Distribution of User Sessions into QoS Levels - Drop Period = 5 Seconds

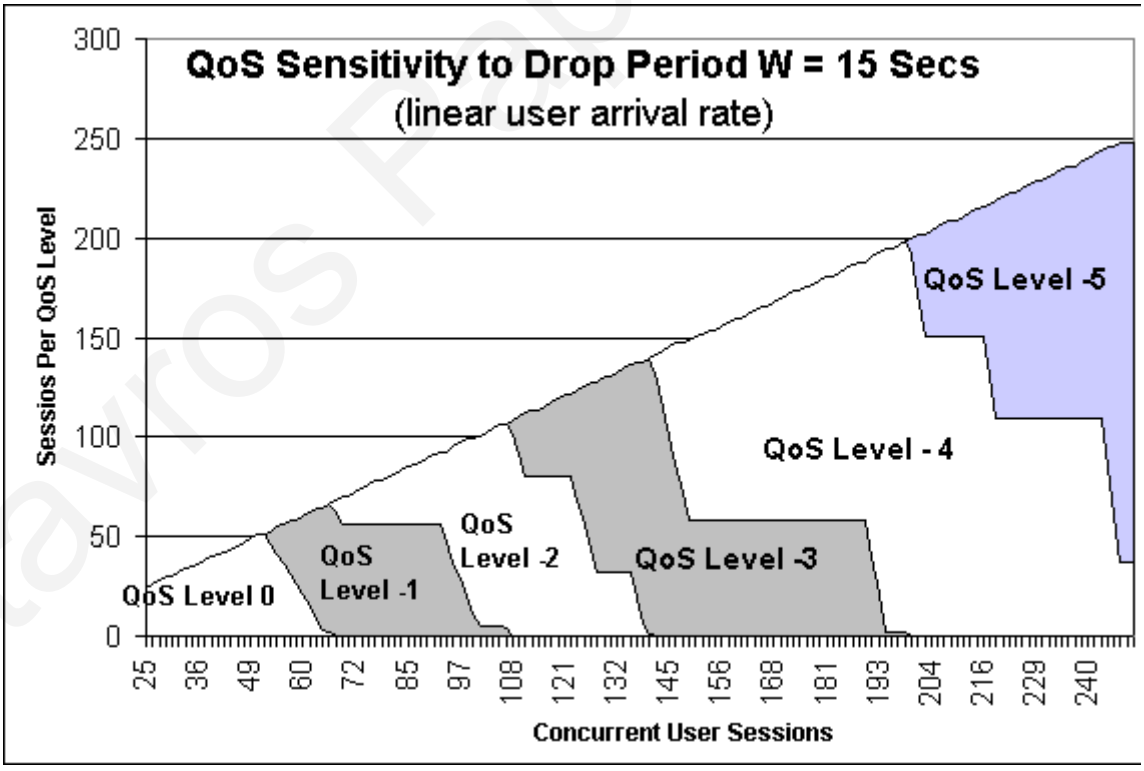

Figure 50: Distribution of User Sessions into QoS Levels - Drop Period = 15 Seconds

Stavros Papastavrou

Parameter	Value
UserDiff	*20%, 60%, 100%*
QoS Check Period W	10secs
Workload Range	100 to 150 Users
Workload Increment	Bursty
User Arrival Rate	50 within 10secs
QoS Threshold	200ms

Table 5: Parameters for the Experiment on QoS Sensitivity to UserDiff - Bursty

6.2.4 QoS Sensitivity to UserDiff - Bursty

Our next two experiments evaluate the QoS-centric Control Scheme under a bursty arrival episode of new user sessions. In the first experiment, we test the system's sensitivity to the number of dropped users per period W, UserDiff. We run the same experiment three times with UserDiff values equal to 20%, 60% and 100%, respectively. For this experiment, the period W is fixed at ten seconds.

We start the experiment with 100 loaded user sessions in Local QoS Level equal to -3. At this workload, the average response time is below the QoS Threshold. We continue by adding 50 new user sessions in the next ten seconds and stop the experiment when the average response time is stabilized below the QoS Threshold. Table 6.2.4 summarizes the parameters for this experiment.

The results of this experiment, plotted in Figure 51, show that the system is more responsive to bursty user arrivals when the UserDiff value is higher. In the experiment, a system with UserDiff value of 100% drops to average response time below the QoS Threshold in about ten seconds once all 50 users are added. On the other hand, a system with a lower UserDiff value not only requires considerably more time to stabilize response times but also suffers from increased response times. This is due to the slow pace of dropping users to lower QoS levels, as already discussed in the experiment of Section 6.2.2 using linear arrival of users.

Stavros Papastavrou

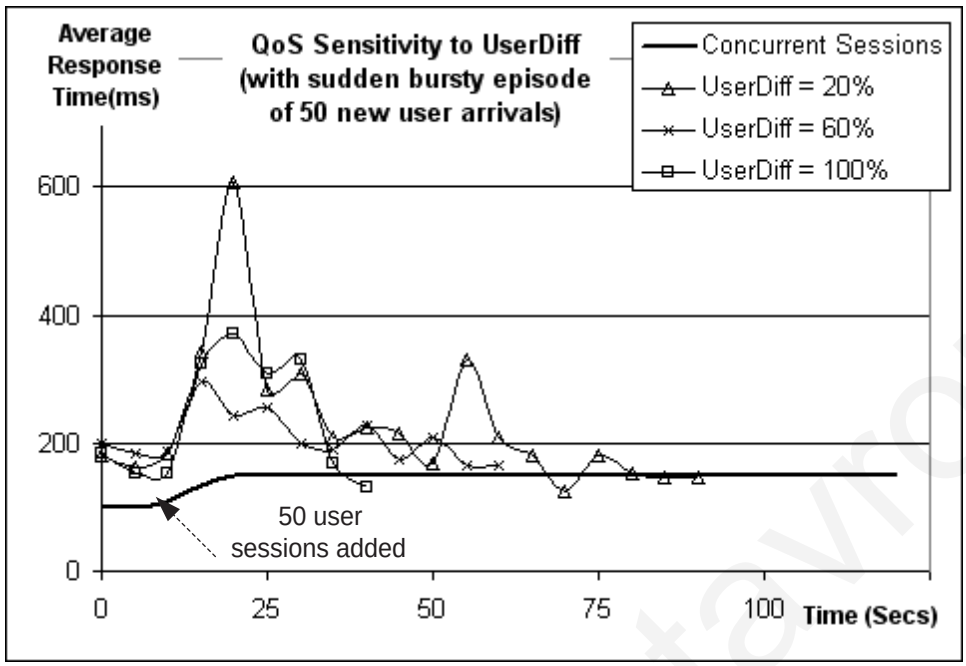

Figure 51: QoS Sensitivity to UserDiff with a Bursty Episode of 50 new User Sessions.

Parameter	Value
QoS Check Period W	*5secs, 10secs, 15secs*
UserDiff	60%
Workload Range	100 to 150 Users
Workload Increment	Bursty
User Arrival Rate	50 within 10secs
QoS Threshold	200ms

Table 6: Parameters for the Experiment on QoS Sensitivity to Drop Period W - Bursty

6.2.5 QoS Sensitivity to Drop Period W - Bursty

In this next experiment, we test the system's sensitivity to the length of the drop period W. We run the same experiment three times with a drop period W of 5, 10 and 15 seconds. The UserDiff is fixed at 60%. Table 6.2.5 summarizes the experimental parameters.

The experiment produced some very interesting results which are plotted in Figure 52. A more lengthy check period leads to a slower initial response to the bursty arrival of users. However, it stabilizes response times faster due to the fact that more users are dropped before the next check.

Stavros Papastavrou

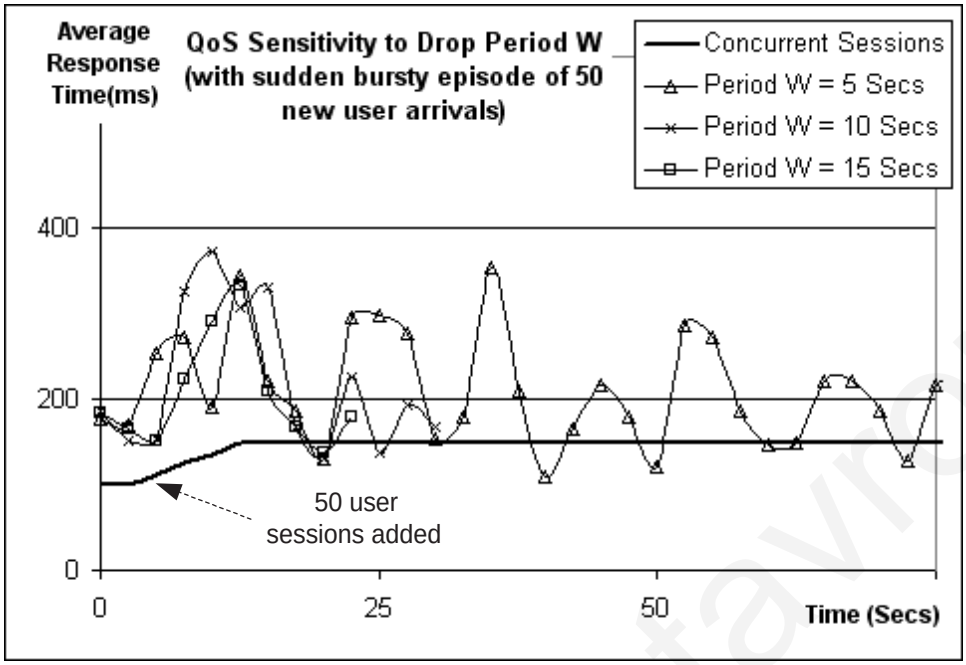

Figure 52: QoS Sensitivity to Drop Period W with a Bursty Episode of 50 new User Sessions.

For example, with UserDiff set to 60%, a check period of 15 seconds guarantees that 60% of users are dropped once within that period since every user submits a request approximately every ten seconds.

On the other hand, a short check period identifies the bursty arrival of users earlier and starts to responds quicker. However, it leads to lengthy periodic increases and decreases of the average response time until that is stabilized below the QoS Threshold. This is because a shorter check period drops only a fraction of user sessions before the next check and, as a result, response times are only momentarily pushed below the QoS Threshold. Consecutive requests from users that were not dropped push again the average response time above the threshold.

For example, with UserDiff at 60%, a system with a check period of five seconds drops only about the half of the 60% of user sessions in each period. However, the rest of the user sessions that were not yet dropped and are due to submit their next request and eventually, when those users do, the average response time is pushed again momentarily higher.

Stavros Papastavrou

6.2.6 Conclusions from the QoS Sensitivity Evaluation

The sensitivity evaluation of the QoS-centric Control Scheme has shown that our QoS Module is capable of maintaining the average response time of user requests below a specific threshold. The sensitivity analysis on UserDiff and check period W has shown that bursty user arrival rates require an aggressive UserDiff value and, unexpectedly, a lengthier check period W. On the other hand, linear user arrival rates do not require an aggressive UserDiff value. In addition, the overall results of using various values for the check period W do not have great difference. Our results point out that a commercial system, which will employ our approach, requires an additional adaptive module to monitor the user arrival rates in order to adjust the check period W and UserDiff parameters accordingly.

6.3 Evaluation of the QLS Algorithm

6.3.1 Overview

Having established that our QoS Module regulates performance, in this section we evaluate our QLS materialization algorithm against the traditional QoD approach. For our experiments, we have not implemented an adaptive module to monitor user arrival rates in order to dynamically adjust the check period W and UserDiff. Instead, we use a linear rate of user arrivals because we are only interested in examining how data quality is affected by the number of concurrent user sessions. For this reason, we use a low UserDiff value of 20% that has proven to maintain the average response time below the QoS Threshold without being aggressive (see Section 6.2.2). In addition, we use a check period W of ten seconds which, in our experiments in Section 6.2.3, was found to be effective without being aggressive.

Stavros Papastavrou

The traditional QoD approach differs from our QLS algorithm at the point when a materialization plan that is picked for materialization (line 32 of Figure 40). Instead of picking up a materialization plan that maximizes QoL, the selected plan maximizes the traditional metric of QoD. For this reason, we modified the existing templates of the bookstore application as follows: (a) we removed the set-view dependency tags and (b) we replaced the link dependency tags with QoD importance weight tags (as discussed in Section 4.2).

For fairness, the QoD importance weight assigned to each fragment is analogous to the average of the importance weights of its link dependencies. For example, a fragment that has 2 link dependencies with weights 40% and 20%, respectively, is given a QoD importance weight of 30%. Finally, all fragment QoD importance weights for each template are normalized to sum up to 100%.

6.3.2 Results on the QLS Algorithm Vs. Traditional QoD

Using the above metrics, our first set of experiments compares our QLS algorithm to the traditional QoD approach. The comparison is on the percentage of pages served with broken link dependencies. A served page contains a broken link dependency when the following two conditions hold: First, a specific fragment with a link dependency to the next template in the users request sequence is not materialized (instead is served from cache). Second, that specific fragment is actually required by the user for navigating to the next template (as discussed in Section 5.3.1).

The results of the experiment, plotted in Figure 53, show that the percentage of pages with broken link dependencies is analogous to the workload. This is because increased workload implies that more users are dropped toward lower QoS levels, as discussed in Section 6.2. As a consequence, more fragments are served from cache which implies that more outdated links are contained in the served pages.

Stavros Papastavrou

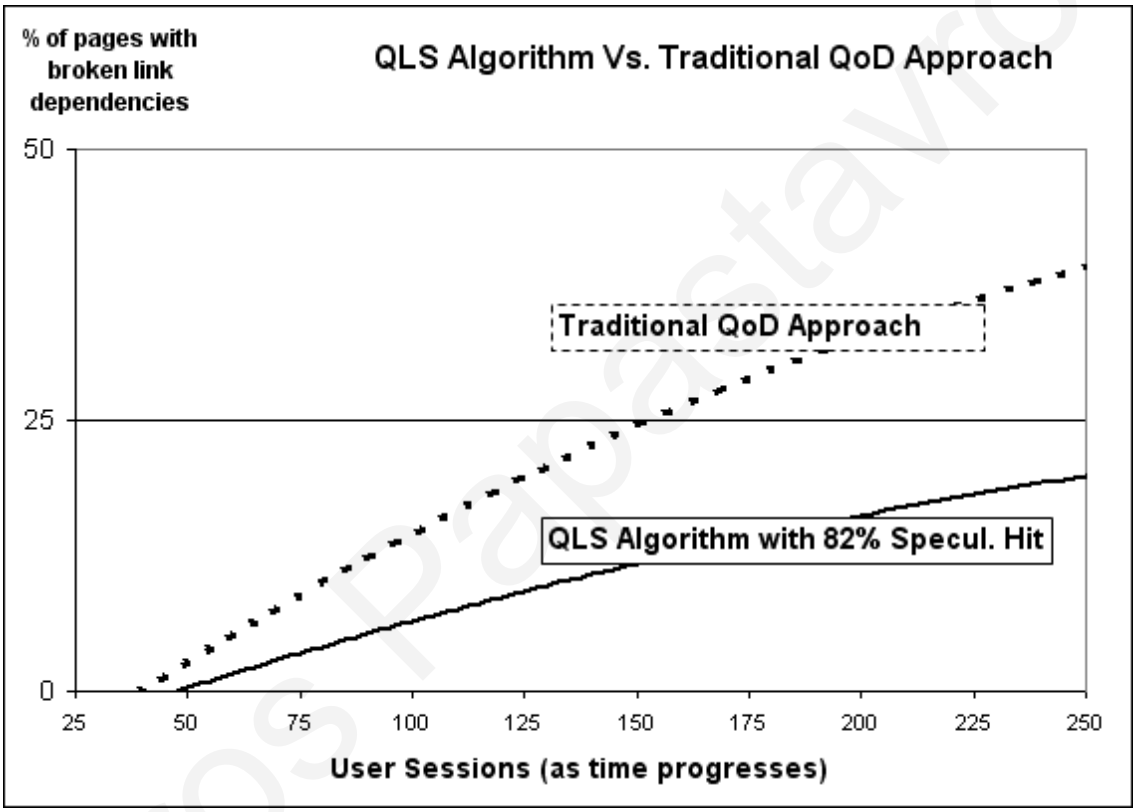

Figure 53: QLS Vs. QoD on Broken Link Dependencies

Stavros Papastavrou

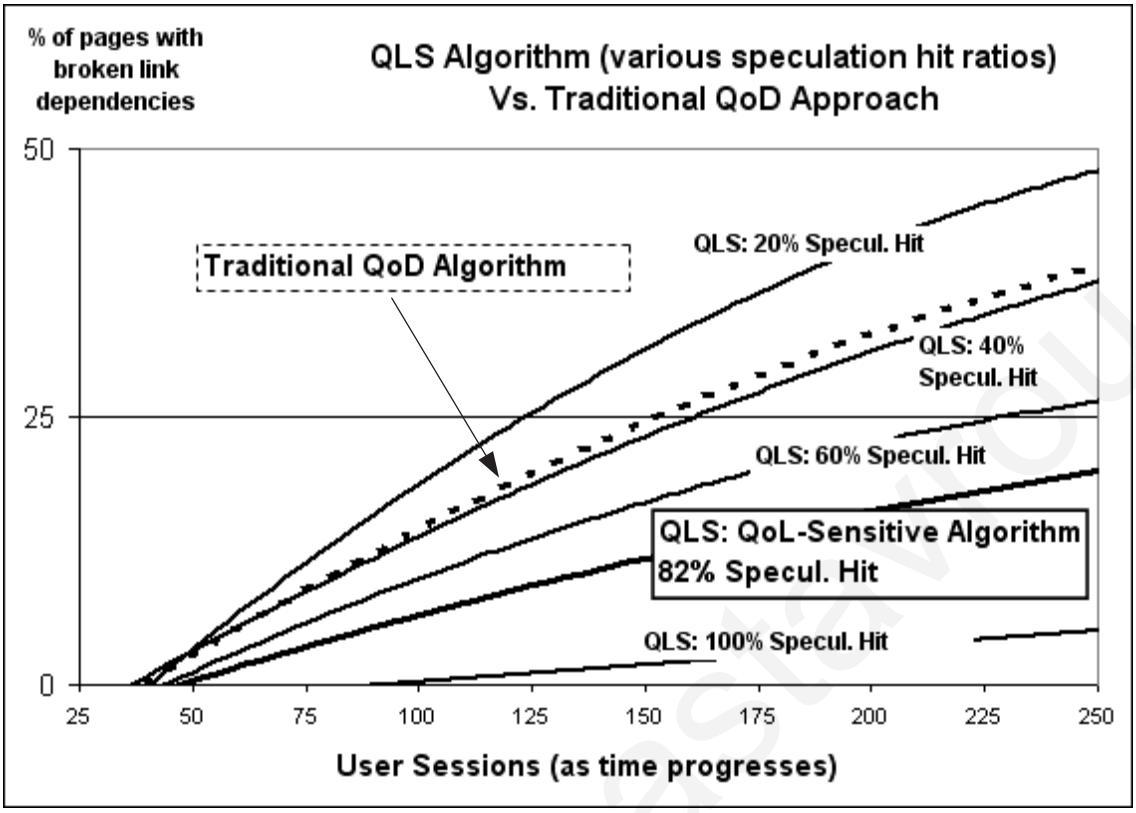

Figure 54: QoL (various speculation hit ratios) Vs. QoD

Moreover, the results in Figure 53 clearly state that the QLS algorithm generates approximately 50% less pages with broken link dependencies than the traditional QoD approach, even at high workload. This is because the QLS algorithm selects the fragments for materialization with link dependencies to the next speculated template of the user. Our analysis has shown that the Speculation Profiling Module used by QLS has a hit ration of 82% in speculating correctly the next template that a user will request.

6.3.3 Impact of Speculation Hit Ratio

To understand the impact of the speculation hit ratio in the previous results, we test again the QLS algorithm by manually altering this ratio. We make four runs of QLS, each one by using 100%, 60%, 40% and 20% speculation hit ratios, respectively. The results, plotted in Figure 54,

Stavros Papastavrou

show that our QLS algorithm still outperforms the traditional QoD approach even when the speculation hit ratio is 50%, in other words, by making one correct and one incorrect speculation. Another interesting observation is that, convergence between our QLS and the traditional QoD approach on the percentage of pages with broken link dependencies occurs when the speculation hit ratio of QLS is approximately just below 40%.

Notice that, even in the ideal case of having a speculation hit ratio of 100%, the QLS algorithm still generates a small percentage of pages with broken link dependencies (about 5%) when the system is overloaded. This is because the heavy workload forces the QLS algorithm to reuse fragments from cache when the next template is speculated correctly.

The significance of these results is that our approach does not solely rely on a good speculation on the next template that a user will request. In addition, the results point out that there is still room for improvement by incorporating better speculation mechanisms. As discussed in Section 4.3.3 and examined in the QLS algorithm presentation in Section 5.1.4, our approach is designed in such a way so that future speculation mechanisms can be easily incorporated.

6.3.4 Results on Server Throughput and Max Concurrent Sessions

In order to understand the impact of the improvements that our QLS algorithm achieves, we run a second set of experiments to discover the "industrial potential" of our approach over the traditional QoD approach. This time, we measure the maximum throughput and maximum number of concurrent user sessions that can be sustained when a threshold of 200ms on QoS is imposed and without fixing the speculation hit ratio.

This experiment differs from the previous ones, since it provides support for handling invalid links in cached fragments. To implement this, we alter the normal request sequence of a user when a template with a cached fragment containing a needed link is received. When this occurs, the user

Stavros Papastavrou

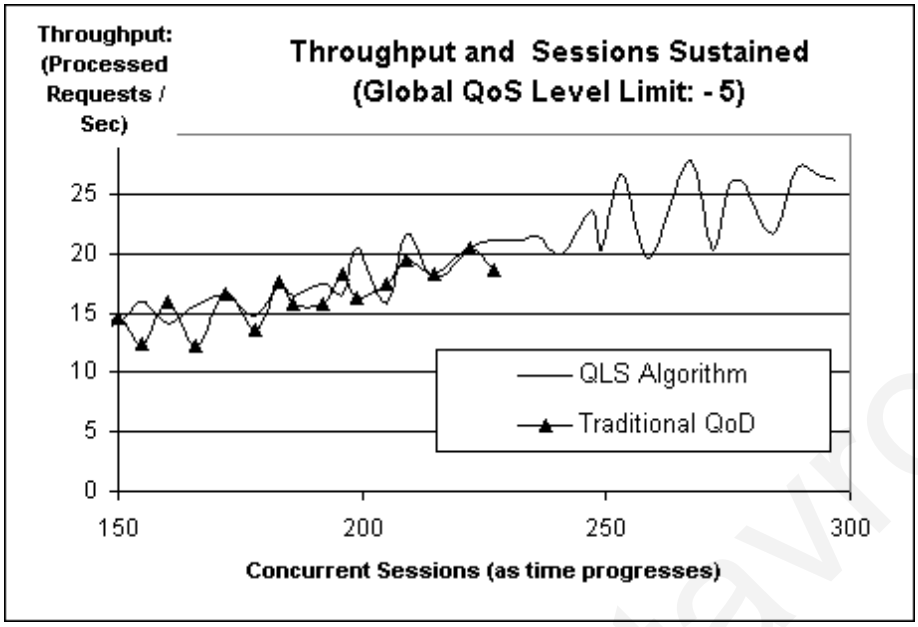

Figure 55: Gains on Throughput and Sessions of the QLS Algorithm

issues an extra special HTTP GET special request to the server *in order to receive only the missing fragment* containing the valid link. Subsequently, the user resumes its template request sequence.

The results of this experiment, plotted in Figure 55, show that the QLS achieves higher throughput by sustaining approximately 25% more concurrent users. This is attributed to difference in increased load at the server to handle the special HTTP GET request issued by users for missing fragments. More specifically, with our approach 50% less special requests issued to the server by users for missing fragments. This is because our approach generates 50% less pages with broken link dependencies, as already discussed in the previous experiment and plotted in Figure 53.

To summarize our results, 82% correct usage plan speculations resulted in better fragment selection by the QLS algorithm, which in turn resulted in 50% less pages with missing links to the next template. Therefore, 50% less special requests for materializing fragments were issued on the server load. As a consequence, the server achieved about 30% more throughput and sustained

Stavros Papastavrou

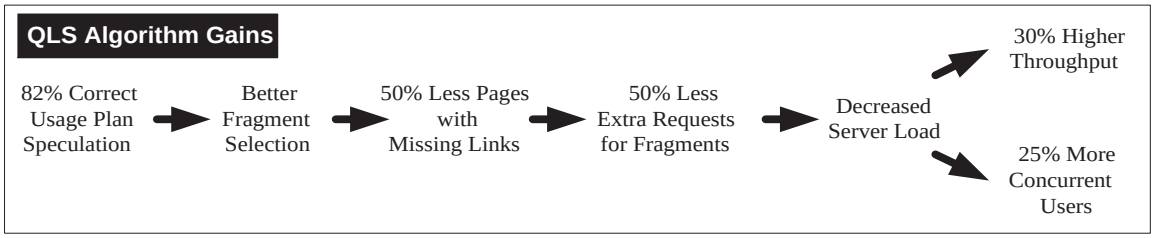

Figure 56: Explaining the Gains of the QLS Algorithm

about 25% more concurrent users. A graphical reasoning on the gains of the QLS algorithm is presented in Figure 56.

6.4 Evaluation of the QoSV Variation of QLS

6.4.1 Results on the Quality of Set-View

In this set of experiments, we compare the QoSV variation of the QLS algorithm to the traditional QoD approach. First, we compare the two approaches on the percentage of unsatisfied set-view dependencies. That is pairs of set-view dependent fragments that are served unsynchronized, as discussed in Section 4.2.3. Then we compare them on the percentage of broken link dependencies. For these experiments, we run the QoSV variation of QLS four times with relax factors for QoL equal to 0%, 10%, 20% and 30%, respectively. Recall that, the relax factor reduces the maximum possible QoL of materialization plans in order for the algorithm to select the plan with the maximum possible QoSV value (see Section 5.2).

The results, shown in Figure 57, reveal the clear gains of the QoSV variation of QLS over the traditional QoD approach. As workload increases pushing more users in lower QoS levels, the number of unsatisfied set-view dependencies nears 100% for the traditional QoD algorithm that has no related provision whatsoever. Instead, our QoSV variation of QLS generates significantly less unsatisfied view dependencies.

Stavros Papastavrou

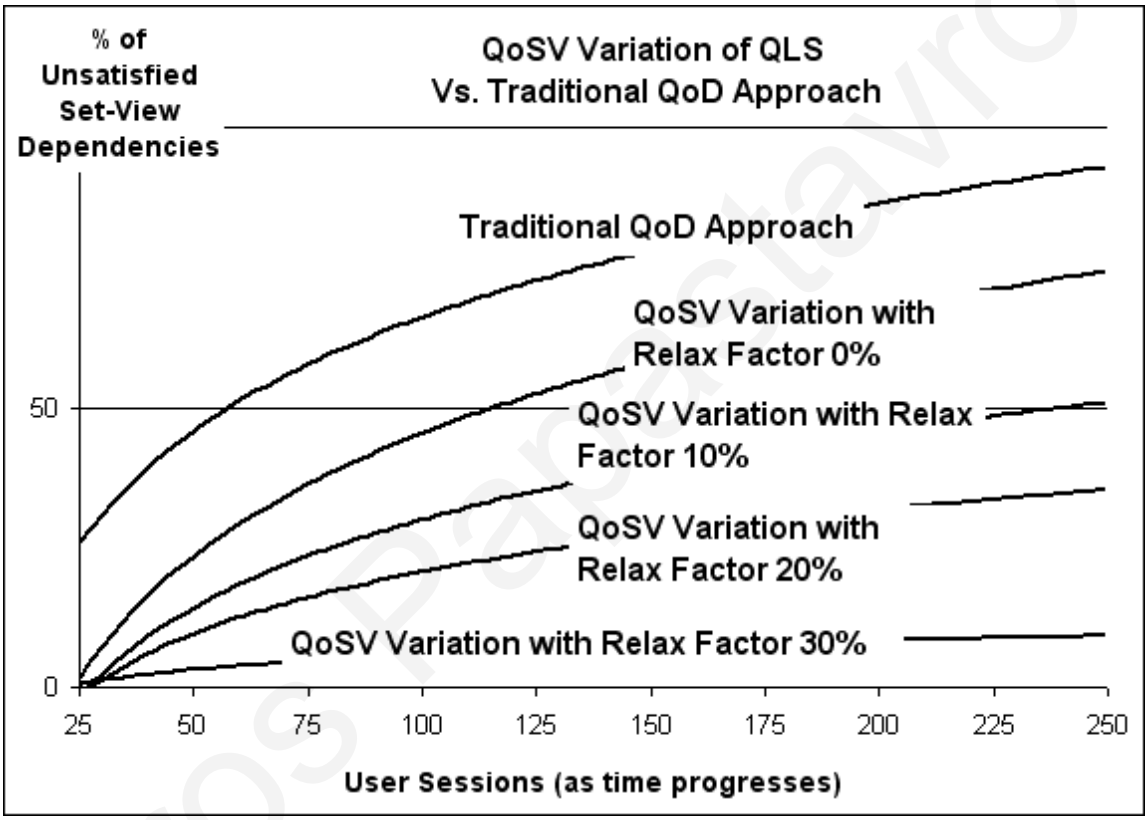

Figure 57: Gains on QoSV Deriving from the QoSV Variation of QLS

Stavros Papastavrou

An immediate observation deriving form the graph presented in Figure 57 is that the gains are significantly better for higher QoL relax factors. This is because a higher QoL relax factor enables a bigger set of candidate materialization plans. From this set, the QoSV variation of QLS chooses the plan with the highest QoSV index value and, therefore, with the fewer unsatisfied view dependencies.

In more detail, for a relax factor of 30% the number of unsatisfied set-view dependencies is about 10% which is an order of magnitude improvement compared to the traditional QoD approach. A 20% relax factor produces about 30% unsatisfied set-view dependencies while a 10% relax factor produces about 50%.

It is worth noticing that even a 0% relax factor produces less unsatisfied set-view dependencies (about 75%). This is because, even though a materialization plan with the highest possible QoL value is selected, still the algorithm picks the plan that maximizes QoSV value. For example, if two materialization plans have the same maximum QoL value, then the algorithm picks the one with the highest QoSV value.

6.4.2 Results on the Quality of Link

The previous experiments have shown that the introduction of the QoL relax factor results in improving the set-wise consistency of content served. However, those gains come at a cost in terms of percentage of pages with broken link dependencies. This is due to reduction on the maximum expected QoL of materialization plans imposed by the QoL relax factor.

As presented in Figure 58, the percentage of pages with broken link dependencies increases for higher QoL relax factors. For a relax factor of 30%, the gains on QoL of the QoSV variation of QLS are reduced to half, compared to the QLS. In order to statistically prove that the enforcement of QoSV compromises the QoL of content served, we performed Paired Samples T-tests on our

Stavros Papastavrou

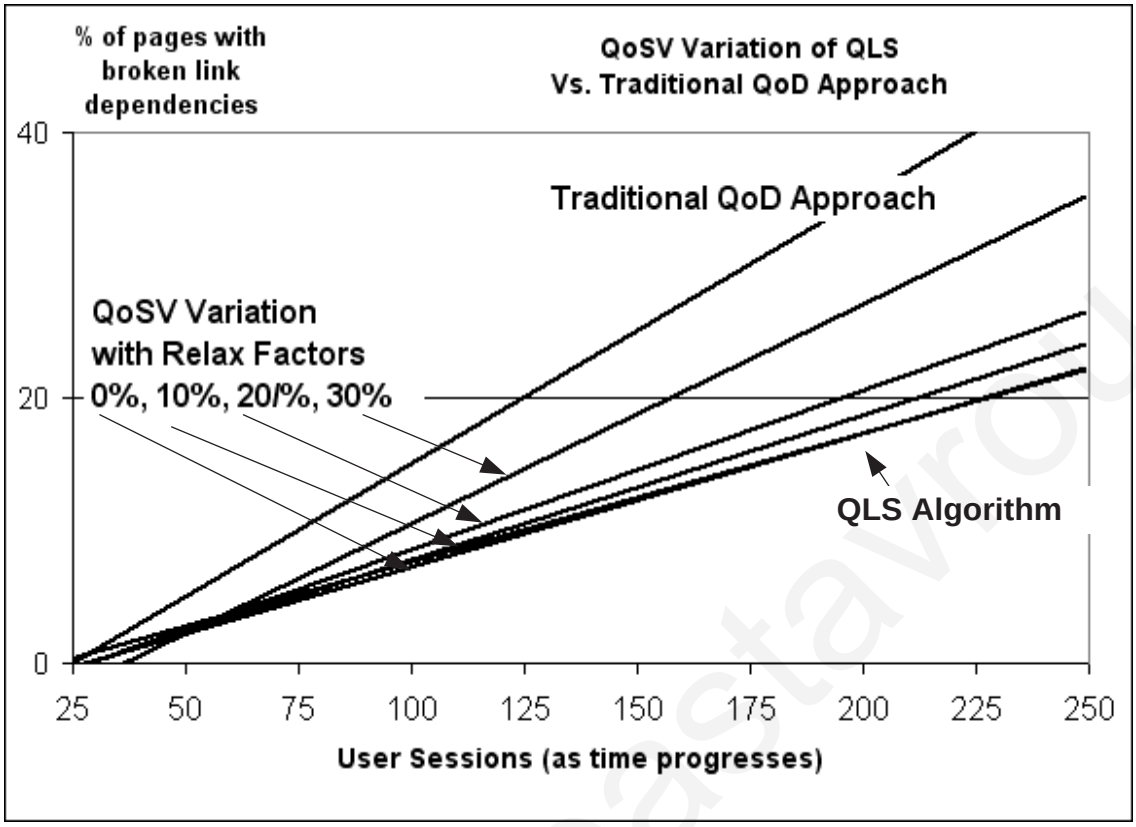

Figure 58: Negative Effect on QoL by the QoSV Variation of QLS

corresponding results [87]. The tests demonstrated that QoL and QoSV of content served at any given time are significantly related.

6.4.3 Results on Server Throughput and Max Concurrent Sessions

Similarly to Section 6.3.4, we run the same setup of experiments to extract the throughput and number of concurrent users that the QoSV variation of the QLS algorithm can sustain. The results are presented in Figure 59, along with the results of Section 6.3.4. The results reveal that the QoSV variation of QLS has lower performance than the original QLS algorithm in terms of throughput and maximum concurrent user sessions. This performance is significantly lower when the algorithm is run for higher relax factors on QoL. However, even at the worst case, in which the

Stavros Papastavrou

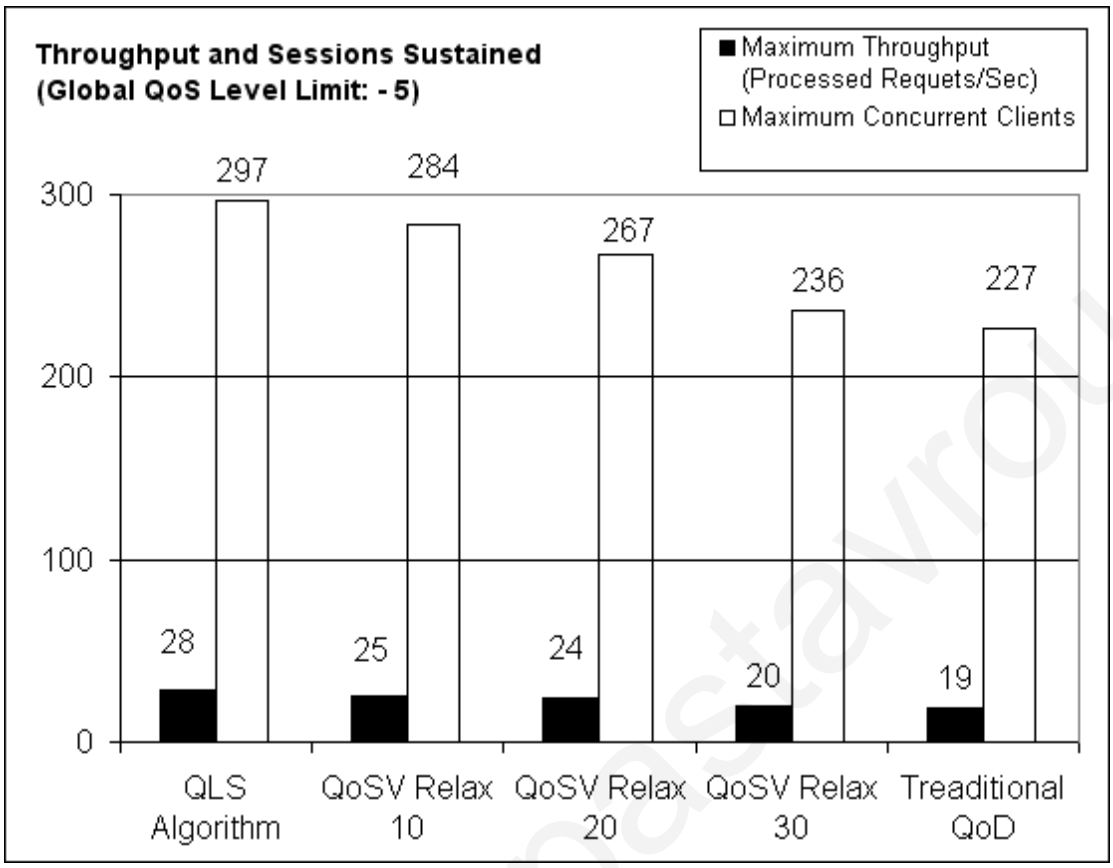

Figure 59: Effect on Throughput and Maximum Concurrent Sessions of the QoSV Variations of QLS

QoSV variation of QLS is run with a relax factor of 30% on QoL, performance is still better than the traditional QoD approach.

Despite the fact that the QoSV variation of QLS causes dramatic improvements on content quality in terms of set-view dependencies, it achieves lower throughput and supports fewer concurrent user sessions. This is because the QoSV sessions generates more pages with broken link dependencies than the QLS, but still, less than the QoD approach. In other words, the promotion of QoSV comes to the expense of QoL, which in turn, decreases the overall system performance in terms of throughput and maximum concurrent user sessions. Similarly to the QLS algorithm, we give a graphical explanation on the gains and compromises of the QoSV variation of the QLS algorithm in Figure 60.

Stavros Papastavrou

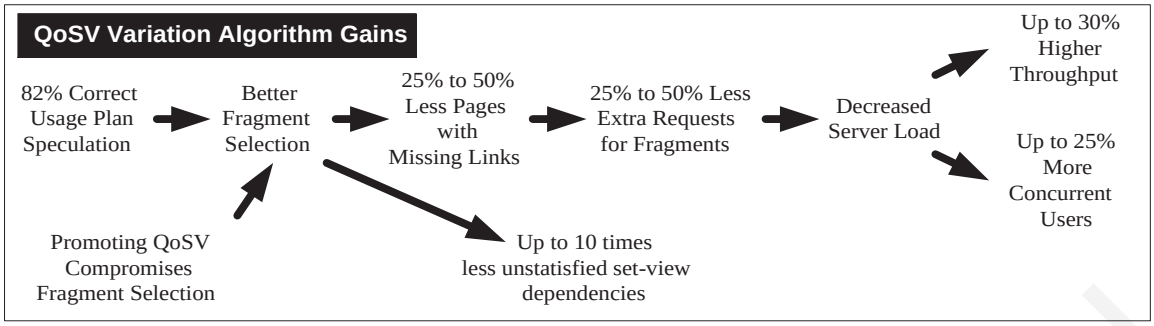

Figure 60: Explaining the Gains and Compromises of the QoSV Variation

6.5 Sensitivity of QoD to Set-View Dependencies

We run a final set of experiments to explore the relation between the traditional metric of QoD and set-view dependencies on fragments, in the absence of link dependencies. This relation is essential to applications where navigation is fixed (implemented with static links), or to applications where a single template has a very high popularity as opposed to other templates. Examples are database-driven web applications (see Section 2.2.1) according to which content is materialized on data updates and not in response to the URL link parameters submitted by users on every request [66]. In this context, which was the one assumed by the tradition QoS-QoD balancing schemes, there are no link dependencies between fragments and templates. There are, however, QoD importance weights on fragments as well as set-view dependencies between them.

In order to run the experiment, we modified the QoSV variation of the QLS algorithm so that QoL importance weights emulate QoD importance weights and we kept the set-view dependencies between fragments. In this way, we were able to use the relax factor on QoL to simulate a relax factor on QoD. This enabled us to measure the effect that the relaxing of QoD has on the percentage of unsatisfied set-view dependencies.

The results of the experiment, plotted in Figure 61, show that the percentage of unsatisfied set-view dependencies nears 100% when there is no relax on QoD, that is, when the maximum

Stavros Papastavrou

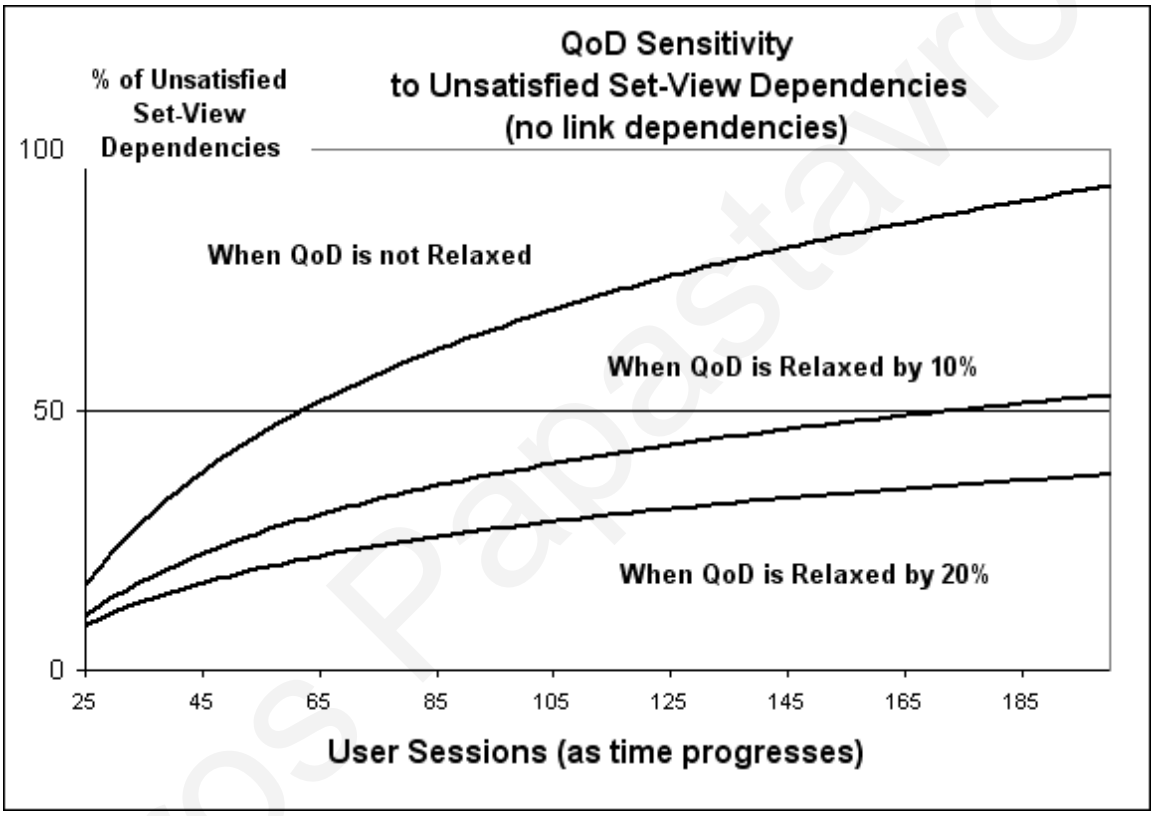

Figure 61: QoSV Vs. QoD

Stavros Papastavrou

number of fragments are materialized for a given QoS (workload) while considering the once with the highest QOD weight.

However, the percentage of unsatisfied set-view dependencies is significantly lower when QoD is relaxed by 10%. This is because a relax factor on QoD allows for more flexibility in choosing which fragments to materialize by ignoring their QoD weight so that more set-view dependencies are satisfied. This precisely is the flexibility that the QoSV variation of QLS is using in order to maximize the number of set-view dependencies during a materialization. This experiment shows that the QoSV variation of QLS is more effective than the traditional QoD approach in supporting the web database applications where links are not important and for which the traditional QoS-QoD balancing scheme was proposed.

6.6 Conclusions on the Experiments of QLS and the QoSV Variation of QLS

Our experiments clearly show the superiority of our algorithms compared to the traditional QoD approach. Below, based on our experimental results, we suggest the space of the applicability of our algorithms.

Our QLS algorithm is more appropriate for web database applications that are characterized by frequent user navigation between templates. Applications that fall into this category are on-line shopping, ticket booking etc. In this cases, the user behavior is characterized by frequent links between templates in order to achieve her/his goal. Side content, such as suggestions and related buys/offers are not of great importance to the user.

On the other hand, the QoSV variation of our QLS algorithm is better suited for applications that are characterized by higher requirements for set-view dependent content. An example application is an on-line bidding or auction web site that provides auctioning tools and displays current results in a single template. In this case, the user does not navigate very often to other

Stavros Papastavrou

templates and so the QoL relax factor should be set relatively higher to promote QoSV of content (for example 30%).

Finally, in our last experiment we demonstrated how our approach can be modified to suit web database applications that do not have any linking requirements between templates (static transitions) but are characterized by strong requirements in keeping consistent content synchronized. Such an application is a stock reporting web site whose content is updated due to database updates.

6.7 Chapter Summary

Our algorithms were compared to the traditional QoD approach. The QLS algorithm serves about 50% less dynamic web pages with missing link dependencies, even at high workload. The QoSV variation of the QLS algorithm serves less pages with unsatisfied set-view dependencies (up to an order of magnitude). Those gains, however, compromise the gains of the QLS algorithm.

The QLS algorithm and its QoSV variation achieve higher server throughput and can sustain more concurrent user sessions than the traditional QoD approach. The QLS is better in that respect, since it serves less content with missing link dependencies.

The gains of our algorithms were achieved by using a profiling mechanism with a very basic usage plan speculation, which achieved a correct speculation ratio of 82%. As a consequence, our algorithms generate significantly less missing links than the traditional QoD approach. Additional experiments reveal that our algorithms achieve better results even at lower correct speculation rates.

Our last experiment reveals the tradeoff between the traditional QoD metric and the QoSV metric in the absence of link dependencies. The experiment stresses out the difficulty in maintaining pairs of fragments synchronized at high workloads, even with related provision.

Stavros Papastavrou

Based on our results, we have listed the circumstances under which our materialization algorithms are suitable to web database applications.

Stavros Papastavrou

Chapter 7

Conclusions

7.1 Summary and Contributions

7.1.1 Summary

The motivation for the dissertation was the complete lack in provision for handling the materialization of link-dependent and set-view dependent content for user-driven web database applications such as e-commerce. Under increased server workload, this lack resulted in the use of cached content containing invalid/broken links that may cause a user session to stall as well as the presentation of inconsistent content in a dynamic web page. This problem was illustration by analyzing the setup of an on-line bookstore and by exploring related work in balancing Quality of Service (QoS) with Quality of Data (QoD) for web database applications.

This dissertation presented algorithms for dynamic content materialization that balance QoS with QoD in terms of the proposed metrics of Quality of Link (QoL) and Quality of Set-View (QoSV). These two metrics substitute the traditional metric of Quality of Data (QoD) and characterize link-dependent and set-view dependent content. The evaluation experiments revealed

Stavros Papastavrou

performance gains up to 25% in terms for server throughput and the maximum number of concurrent user sessions, while maintaining high levels of navigation ability and data consistency as opposed to the traditional QoD approaches.

7.1.2 Contributions

The contributions of this dissertation are the following:

- The first contribution was the identification and classification of *content dependencies*. We identified *generation dependency*, according to which non-materialized fragments that are sources of *generation dependencies* prevent their dependent fragments from materializing. We identified set-view dependency, according to which two dependent fragments must be both materialized (or reused from cache with synchronous versions) during template processing so that they present consistent content. We identified *link dependency*, according to which a fragment contains HTML links that enable the user to navigate to another template.

- *Quality of link* (QoL) was introduced as a new data quality metric that quantifies the existence of freshly materialized fragments inside a template with link dependencies to another template. In other words, it measures the ability of a user to navigate to the next page from within its current page.

- *Quality of set-view* (QoSV) was introduced as a metric that quantifies the set-wise consistency of fragments inside a template. In other words, it measures the degree in which set-view dependent fragments are synchronized inside a template.

- The fourth contribution of this dissertation was the introduction of *Usage Plans* (UP). Their purpose is to capture recurrent patterns in the sequence of user requests such as consecutive

Stavros Papastavrou

requests on a specific template or consecutive requests between two templates. Usage Plans were employed for speculating on the next template that a user will request.

- The main practical contribution of this dissertation was the development of two algorithms for content materialization. The first one, the *QLS algorithm*, optimizes for link dependencies. Its goal is to balance QoS with QoL in order to enable seamless user navigation, even at high workloads, in which more fragments are reused from cache and contain outdated HTML links. The second one is a variation of the QLS and its called QoSV variation. Its additional goal is to promote set-view consistency of materialized fragments by satisfying their set-view dependencies. Both algorithms take as input the current server workload, the setup of a web site in terms of content and link dependencies, the list of usage plans for the web site and the expected user behavior facilitated by a basic profiling mechanism.

  Both algorithms were implemented and evaluated on an experimental platform that runs a bookstore web database application. Our implementation was not a simulation, but an emulation of an on-line Bookstore application. The major difference from a real-world bookstore application was that we did not render the HTML content for commercial use.

- Finally, two minor contributions are (a) the taxonomy of web database applications and (b) the CFP framework as a means of classifying related research on enhancing QoS for web database applications.

7.2 Applications

In this section, we explain how our approach can benefit existing applications.

Stavros Papastavrou

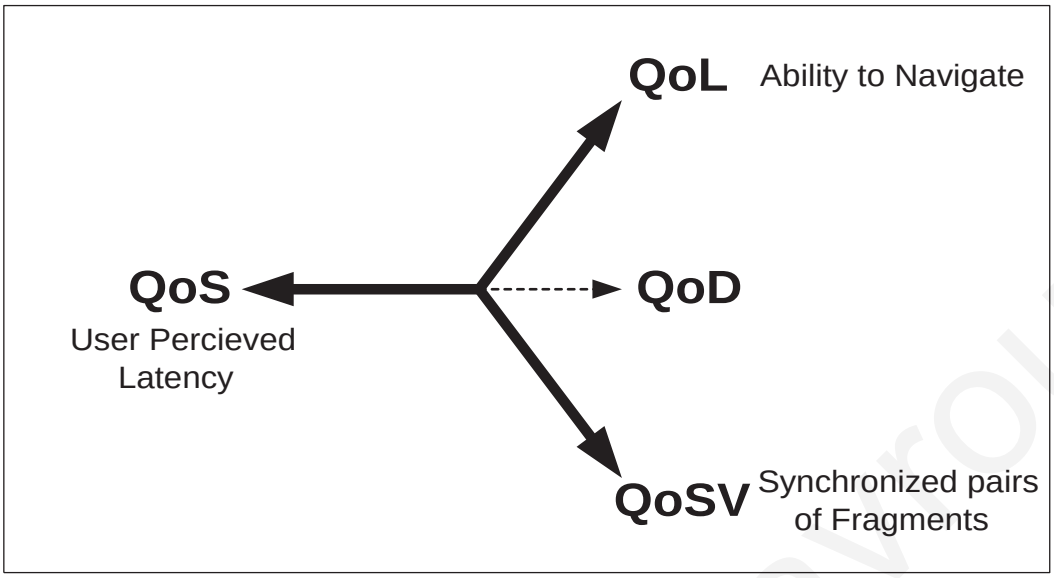

Figure 62: Tradeoff Between QoS, QoL and QoSV

7.2.1 Broad Impact

Our approach can benefit e-commerce vendors that seek to boost performance with less hardware under higher workloads. As the experimental results have revealed, our approach can sustain approximately 25% more concurrent user sessions and increase throughput accordingly.

Our approach requires the installation of an application server with related support. This is the same case with popular industrial-strength applications servers, such as ColdFusion, ASP.NET and PHP. In addition, a necessary on-time off-line setup is required for fragment and dependency weights tagging, as well as defining the list of usage plans between the dynamic web pages of the application. The bridging between the web server and the application server, supporting our approach, is achieved with mapping, on the web server, the file extension of our template files to the application server. In addition, the application server uses language and database-specific connection with the application database. In our case, we connect our Java-based application server with the Microsoft SQL Server application database using a JDBC Type 4 driver [89].

Stavros Papastavrou

Our main contributions target primarily web database applications that fall in to the category of e-commerce such as an on-line Bookstore. However, database-driven web applications, such as stock-related information web sites, can benefit with the employment of the QoSV metric, in the absence of link dependencies. To this end, the results of the experiment in Section 6.5 suggest that related provision is required for maintaining pairs of set-view dependent fragments synchronized under heavy workload. This provision is implemented by the QoSV variation of QLS.

7.2.2 Mobile Content Adaptation

Our approach can also facilitate the adaptation of content on mobile devices. The basic principle behind current approaches is the manipulation of already materialized content to fit the small screen of mobile devices and to minimize the size of transmitted data [69, 60, 35]. Practices include filtering out advertisements, content that cannot be rendered (i.e., vendor-specific animations such as Flash-based), unnecessary HTML tags, cascading style files, Javascript files and large images.

The approach in [115] extracts the most important parts of large text files using a fractal-based approach. It assumes that the most necessary to the user parts of a text file are those that visually match the layout of a fractal. For example, only the sentences in the beginning, in the middle and at the end of documents are initially sent to the users. If necessary, the procedure is repeated recursively for the two half-parts of the document. In addition, [51] rearranges a materialized web page into content fragments so that the fragments are rendered in a convenient top-down fashion. All the approaches presented above can be implemented either at the server-side, the proxy, or even at the user when bandwidth is not an issue.

Our QLS algorithm can benefit current approaches by transmitting only fragments with link dependencies to the speculated next template of a mobile user. The gains are twofold: unnecessary

Stavros Papastavrou

cached fragments are not transmitted, sparing in this way bandwidth and the filtering of content has significantly less input to process. Missing fragments can be requested on demand by mobile users with links placed at the end of the fragments transmitted.

7.2.3 Differentiated Level of Service

Our approach can be applied to differentiate the level of service for users of, for example, an on-line store. Users that are frequent buyers should enjoy faster response times and fresher content under heavy workload, as opposed to "surfers" that should be transmitted more cached content as slower response times.

Our approach can facilitate differentiated level of service on high workloads by distributing user sessions into QoS levels according to their characterization. This characterization may split user sessions into groups such as "Gold Customers", "Silver Customers" and "Bronze Customers". Figure 63 illustrates the proposed distribution of user into QoS level. At lower workloads, all users are treated equally, receiving only freshly materialized content fragments. As workload increases, some users are inevitably pushed into lower QoS levels, so that the average response time is kept below the threshold. "Gold Customers" are kept into higher QoS levels so that fresh content and links are to their full disposal. In the other hand, "Bronze Customers" are the ones that receive the least fresh content.

In a similar application, our approach can facilitate the distribution of users by response time preferences. Users that are "in a hurry" are always kept into lower QoS levels, according to the overall server workload. In the other hand, users with no response considerations are distributed into higher QoS levels. The application is illustrated in Figure 64.

Stavros Papastavrou

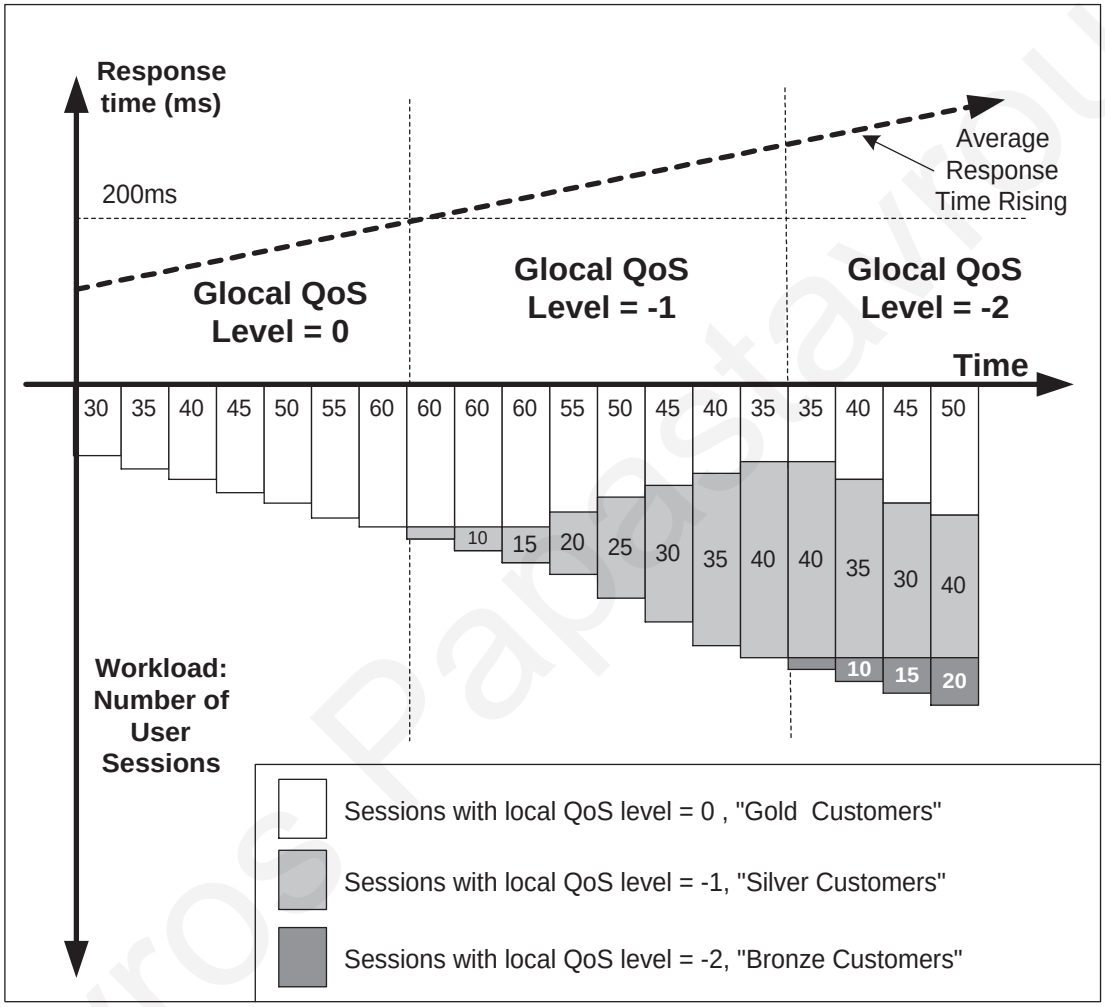

Figure 63: Distribution of User Sessions (Customers) in QoS levels According to their Buying History. At higher workloads, provision is taken so that "better customers" maintain their QoS level higher than "worse customers" and receive only fresh content.

Stavros Papastavrou

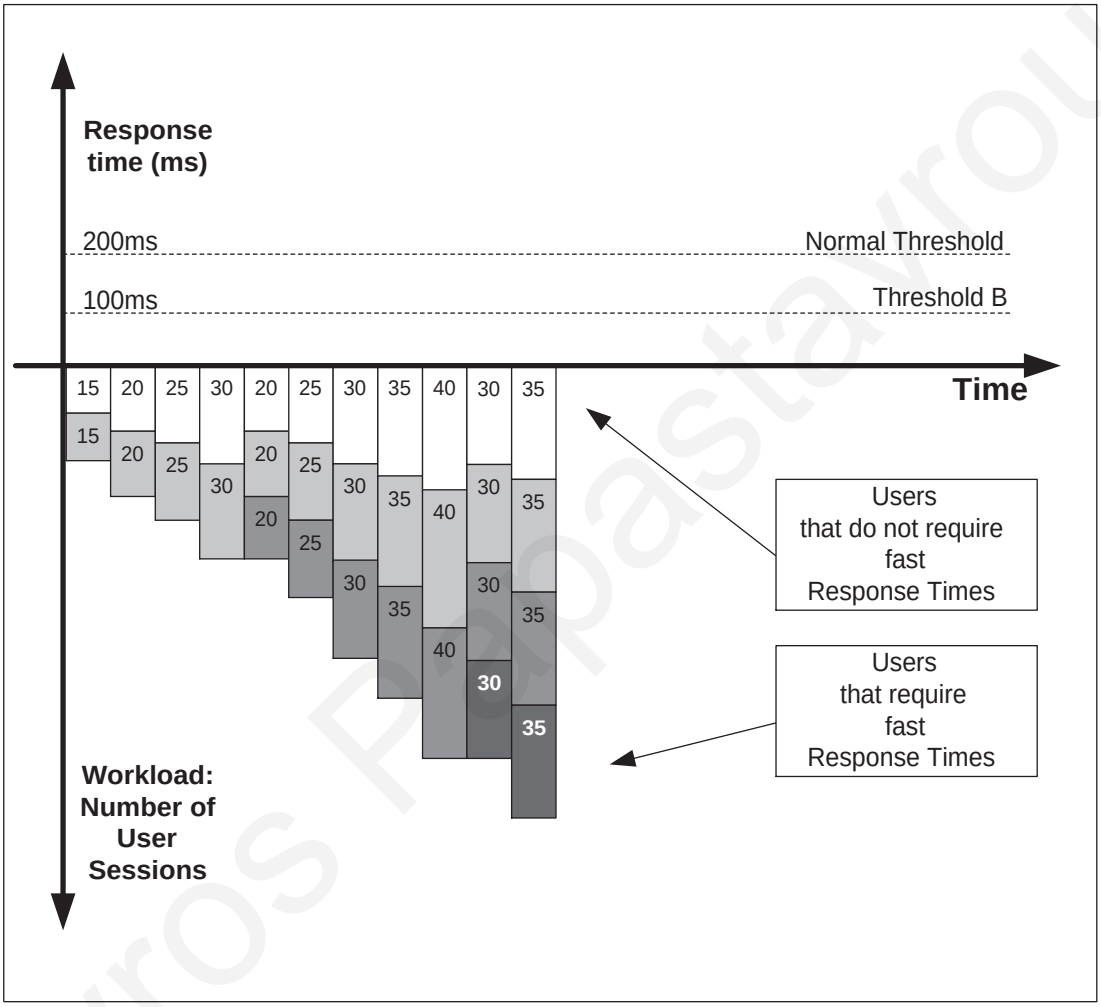

Figure 64: Distribution of User Sessions (Customers) in QoS Levels According to their Response Time Preferences. Users that require fast response times are always kept into lower QoS levels, even at lower workloads and have their own response time threshold (B).

Stavros Papastavrou

To implement differentiated level of service, techniques from user profiling and personalization of content for e-commerce applications [79, 46] can be used. According to [46], web personalization is the customization of a web site to the needs and expectations of specific users. This customization is achieved by taking advantage of the knowledge acquired from the analysis of the user traces or by user-submitted preferences.

7.3 Impact and Future Work

The impact of this dissertation to both the literature and the industry is its contribution towards creating a semantics-based methodology of measuring data quality. We consider quality of content as a multi-faceted metric that relates to user navigation and to set-wise relation of content.

Our future work focuses on providing a better speculation profiling mechanism that to increase the speculation hit ratio. To achieve this, we are building support for on-line adapting the QoL and QoSV fragment importance weights. This adaptation will be user-specific and will be assisted by monitoring the popularity of the fragments themselves.

Stavros Papastavrou

Bibliography

[1] The apache web server. http://www.apache.org.

[2] The edge-side includes initiative. http://www.esi.org.

[3] Fastcgi white paper from open market, inc. http://www.fastcgi.com/devkit/doc/fastcgi-whitepaper/fastcgi.htm.

[4] Ncsa httpd tutorial: Server side includes (ssi). http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html.

[5] The ncsa zeus web server. http://www.zeus.com.

[6] Nielsen online internet media and market research firm. http://www.nielsen-online.com.

[7] Xcache: The cache management solution. http://www.xcache.com.

[8] ABITEBOUL, S., BENJELLOUN, O., MANOLESCU, I., MILO, T., AND WEBER, R. Active xml: Peer-to-peer data and web services integration. In *International Conference on Very Large Data Bases* (2002), pp. 1087–1090.

[9] AGGARWAL, C., WOLF, J. L., AND YU, P. S. Caching on the world wide web. *IEEE Transactions on Knowledge and Data Engineering 11*, 1 (1999), 94–107.

[10] AKOKA, J., BERTI-EQUILLE, L., BOUCELMA, O., BOUZEGHOUB, M., COMYN-WATTIAU, I., COSQUER, M., GOASDOUÍ-THION, V., KEDAD, Z., NUGIER, S., PERALTA, V., AND SISAID-CHERFI, S. A framework for quality evaluation in data integration systems. In *International Conference on Enterprise Information Systems* (2007), pp. 170–175.

[11] AMZA, C., CECCHET, E., CHANDA, A., COX, A., ELNIKETY, S., GIL, R., MARGUERITE, J., RAJAMANI, K., AND ZWAENEPOEL, W. Specification and implementation of dynamic web site benchmarks. In *IEEE 5th Annual Workshop on Workload Characterization* (2002).

[12] ARLITT, M. Characterizing web user sessions. *SIGMETRICS Perform. Eval. Rev. 28*, 2 (2000), 50–63.

[13] ARLITT, M. F., AND WILLIAMSON, C. L. Internet web servers: workload characterization and performance implications. *IEEE/ACM Transactions on Networking 5*, 1 (1997), 631–645.

Stavros Papastavrou

[14] ARLITT, M. F., AND WILLIAMSON, C. L. Internet web servers: workload characterization and performance implications. *IEEE/ACM TON (Transactions on Networking) 5*, 5 (1997), 631–645.

[15] AZAR, Y., FEDER, M., LUBETZKY, E., RAJWAN, D., AND SHULMAN, N. The multicast bandwidth advantage in serving a web site. In *Networked Group Communication* (2001).

[16] BANAVAR, G., CHANDRA, T. D., MUKHERJEE, B., NAGARAJARAO, J., STROM, R. E., AND STURMAN, D. C. An efficient multicast protocol for content-based publish-subscribe systems. In *IEEE International Conference on Distributed Computing System* (1999), pp. 262–272.

[17] BEAVER, J., MORSILLO, N., PRUHS, K., CHRYSANTHIS, P. K., AND LIBERATORE, V. Scalable dissemination: What's hot and what's not. In *WebDB* (2004), pp. 31–36.

[18] BEAVER, J., PRUHS, K., CHRYSANTHIS, P. K., AND LIBERATORE, V. The multicast pull advantage in dissemination-based data delivery. In *Third Hellenic Data Management Symposium* (2004).

[19] BEAVER, J., PRUHS, K., CHRYSANTHIS, P. K., AND LIBERATORE, V. Improving the hybrid data dissemination model of web documents. *World Wide Web 11*, 3 (2008), 313–337.

[20] BERNERS-LEE, T. Uniform resource locators (url). http://www.w3.org/Addressing/rfc1738.txt, 1994.

[21] BERNERS-LEE, T. Hypertext markup language specification 2.0. www.ics.uci.edu/ietf/html/html2spec.ps.gz, 1995.

[22] BERNERS-LEE, T., CAILLIAU, R., AND GROFF, J.-F. The world-wide web. *Computer Networks and ISDN Systems 25*, 4-5 (1992), 454–459.

[23] BERNSTEIN, P., BRODIE, M., CERI, S., DEWITT, D., FRANKLIN, M., GARCIA-MOLINA, H., GRAY, J., HELD, J., HELLERSTEIN, J., JAGADISH, H. V., LESK, M., MAIER, D., NAUGHTON, J., PIRAHESH, H., STONEBRAKER, M., AND ULLMAN, J. The asilomar report on database research. *SIGMOD Record 27*, 4 (1998), 74–80.

[24] BOBROWSKI, M., MARR, M., AND YANKELEVICH, D. A software engineering view of data quality. In *Intl. Workshop on Information Quality in Information Systems* (2004).

[25] BORNHÖVD, C., ALTINEL, M., KRISHNAMURTHY, S., MOHAN, C., PIRAHESH, H., AND REINWALD, B. Dbcache: middle-tier database caching for highly scalable e-business architectures. In *SIGMOD Conference* (2003), pp. 662–662.

[26] BOUAZIZI, E., DUVALLET, C., AND SADEG, B. Management of qos and data freshness in rtdbss using feedback control scheduling and data versions. In *ISORC '05: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (2005), pp. 337–340.

[27] BRAUN, H.-W., AND CLAFFY, K. C. Web traffic characterization: An assesment of the impact of caching documents from ncsa's web server. *Computer Networks and ISDN Systems 28*, 1&2 (1995), 37–51.

Stavros Papastavrou

[28] BRESLAU, L., CAO, P., FAN, L., PHILLIPS, G., AND SHENKER, S. Web caching and zipf-like distributions: Evidence and implications. In *IEEE Conference on Computer Communications* (1999), pp. 126–134.

[29] BRIGHT, L., AND RASCHID, L. Using latency-recency profiles for data delivery on the web. In *International Conference on Very Large Data Bases* (2002), pp. 550–561.

[30] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual web search engine. *Computer Networks 30*, 1-7 (1998), 107–117.

[31] CAO, P., ZHANG, J., AND BEACH, K. Active cache: caching dynamic contents on the web. *Distributed Systems Engineering 6*, 1 (1999), 43–50.

[32] CARNEY, D., LEE, S., AND ZDONIK, S. B. Scalable application-aware data freshening. In *IEEE International Conference on Data Engineering* (2003), pp. 481–492.

[33] CECCHET, E., CHANDA, A., ELNIKETY, S., MARGUERITE, J., AND ZWAENEPOEL, W. Performance comparison of middleware architectures for generating dynamic web content. In *Middleware Conference* (2003), pp. 282–304.

[34] CHALLENGER, J., IYENGAR, A., WITTING, K., FERSTAT, C., AND REED, P. A publishing system for efficiently creating dynamic web content. In *IEEE Conference on Computer Communications* (2000), pp. 844–853.

[35] CHEN, Y., MA, W.-Y., AND ZHANG, H.-J. Detecting web page structure for adaptive viewing on small form factor devices. In *International World Wide Web Conference* (2003), pp. 225–233.

[36] CHO, J., AND GARCIA-MOLINA, H. Synchronizing a database to improve freshness. In *SIGMOD Conference* (May 2000), pp. 117–128.

[37] CHO, J., AND GARCIA-MOLINA, H. Estimating frequency of change. *ACM Trans. Interet Technol. 3*, 3 (2003), 256–290.

[38] CHO, J., AND NTOULAS, A. Effective change detection using sampling. In *International Conference on Very Large Data Bases* (2002), pp. 514–525.

[39] CUNHA, C., BESTAVROS, A., AND CROVELLA, M. Characteristics of www client-based traces. Tech. Rep. TR-95-010, Boston University, 1995.

[40] DATTA, A., DUTTA, K., RAMAMRITHAM, K., THOMAS, H. M., AND VANDERMEER, D. E. Dynamic content acceleration: A caching solution to enable scalable dynamic web page generation. In *SIGMOD Conference* (2001), p. 616.

[41] DATTA, A., DUTTA, K., THOMAS, H. M., VANDERMEER, D. E., RAMAMRITHAM, K., AND FISHMAN, D. A comparative study of alternative middle tier caching solutions to support dynamic web content acceleration. In *International Conference on Very Large Data Bases* (2001), pp. 667–670.

[42] DATTA, A., DUTTA, K., THOMAS, H. M., VANDERMEER, D. E., SURESHA, AND RAMAMRITHAM, K. Proxy-based acceleration of dynamically generated content on the world wide web: An approach and implementation. In *SIGMOD Conference* (2002), pp. 97–108.

Stavros Papastavrou

[43] DINGLE, A., AND PÁRTL, T. Web cache coherence. *Computer Networks 28*, 7-11 (1996), 907–920.

[44] DOLEV, D., MOKRYN, O., SHAVITT, Y., AND SUKHOV, I. An integrated architecture for the scalable delivery of semi-dynamic web content. In *IEEE Symposium on Computers and Communications* (2002).

[45] DOUGLIS, F., HARO, A., AND RABINOVICH, M. HPP: HTML macro-preprocessing to support dynamic document caching. In *USENIX Symposium on Internet Technologies and Systems* (1997).

[46] EIRINAKI, M., AND VAZIRGIANNIS, M. Web mining for web personalization. *ACM Transactions on Interet Technology 3*, 1 (2003), 1–27.

[47] GAL, A. Obsolescent materialized views in query processing of enterprise information systems. In *ACM Conference on Information and Knowledge Management* (1999), pp. 367–374.

[48] GERTZ, M., ÖZSU, M. T., SAAKE, G., AND SATTLER, K.-U. Report on the dagstuhl seminar: "data quality on the web". *SIGMOD Record 33*, 1 (2004), 127–132.

[49] GUNDAVARAM, S. *CGI programming on the World Wide Web*. O'Reilly & Associates, Inc., 1996.

[50] GUPTA, A., AND MUMICK, I. S., Eds. *Materialized views: techniques, implementations, and applications*. MIT Press, Cambridge, MA, USA, 1999.

[51] GUPTA, S., KAISER, G., NEISTADT, D., AND GRIMM, P. Dom-based content extraction of html documents. In *International World Wide Web Conference* (New York, NY, USA, 2003), ACM, pp. 207–214.

[52] HOLMEDAHL, V., SMITH, B., AND YANG, T. Cooperative caching of dynamic content on a distributed web server. In *IEEE International Symposium on High Performance Distributed Computing* (1998), p. 243.

[53] HUANG, Y., SLOAN, R. H., AND WOLFSON, O. Divergence caching in client-server architectures. In *International Conference on Parallel and Distributed Information Systems* (1994), pp. 131–139.

[54] HUNT, J. J., VO, K.-P., AND TICHY, W. F. Delta algorithms an empirical analysis. *ACM Transactions on Software Engineering and Methodology 7*, 2 (1998), 192–214.

[55] IYENGAR, A., AND CHALLENGER, J. Improving web server performance by caching dynamic data. In *USENIX Symposium on Internet Technologies and Systems* (1997).

[56] KIM, M. Building web-integrated database applications (a review of web database applications with php & mysql by hugh e. williams and david lane). *IEEE Distributed Systems Online 4*, 2 (2003).

[57] KRISHNAMURTHY, B., WILLS, C. E., AND ZHANG, Y. On the use and performance of content distribution networks. In *Internet Measurement Workshop* (2001), pp. 169–182.

Stavros Papastavrou

[58] KRISHNAN, P., RAZ, D., AND SHAVITT, Y. The cache location problem. *IEEE/ACM Transactions on Networking 8*, 5 (2000), 568–582.

[59] KRISTOL, D. M. Http cookies: Standards, privacy, and politics. *ACM Trans. Interet Technol. 1*, 2 (2001), 151–198.

[60] LAAKKO, T., AND HILTUNEN, T. Adapting web content to mobile user agents. *IEEE Internet Computing 9*, 2 (2005), 46–53.

[61] LABRINIDIS, A., AND ROUSSOPOULOS, N. On the materialization of webviews. In *WebDB (Informal Proceedings)* (1999), pp. 79–84.

[62] LABRINIDIS, A., AND ROUSSOPOULOS, N. Generating dynamic content at database-backed web servers: cgi-bin vs. mod_perl. *SIGMOD Record 29*, 1 (2000), 26–31.

[63] LABRINIDIS, A., AND ROUSSOPOULOS, N. Webview materialization. *SIGMOD Record 29*, 2 (2000), 367–378.

[64] LABRINIDIS, A., AND ROUSSOPOULOS, N. Update propagation strategies for improving the quality of data on the web. In *International Conference on Very Large Data Bases* (2001), pp. 391–400.

[65] LABRINIDIS, A., AND ROUSSOPOULOS, N. Balancing performance and data freshness in web database servers. In *International Conference on Very Large Data Bases* (2003), pp. 393–404.

[66] LABRINIDIS, A., AND ROUSSOPOULOS, N. Exploring the tradeoff between performance and data freshness in database-driven web servers. *VLDB Journal 13*, 3 (2004), 240–255.

[67] LARSON, P.-A., GOLDSTEIN, J., AND ZHOU, J. Transparent mid-tier database caching in sql server. In *SIGMOD Conference* (2003), pp. 661–661.

[68] LARUS, J. R., AND PARKES, M. Using cohort scheduling to enhance server performance. In *Conference on Languages, Compilers, and Tools for Embedded Systems* (2001), pp. 182–187.

[69] LEE, E., KANG, J., CHOI, J., AND YANG, J. Topic-specific web content adaptation to mobile devices. In *WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence* (2006), pp. 845–848.

[70] LI, W., ZHANG, W., LIBERATORE, V., PENKROT, V., BEAVER, J., SHARAF, M. A., ROYCHOWDHURY, S., CHRYSANTHIS, P. K., AND PRUHS, K. An optimized multicast-based data dissemination middleware. In *IEEE International Conference on Data Engineering* (2003), pp. 762–764.

[71] LI, W.-S., PO, O., HSIUNG, W.-P., CANDAN, K. S., AND AGRAWAL, D. Engineering and hosting adaptive freshness-sensitive web applications on data centers. In *International World Wide Web Conference* (2003), pp. 587–598.

[72] LIU, C., ARMSTRONG, G., LEE, D., AND LU, J. Web survey design with active server pages: a new research method. In *International Conference on Information Resources Management* (2000), pp. 1188–1189.

Stavros Papastavrou

[73] LIU, Y.-H., DANTZIG, P., WU, C.-F. E., CHALLENGER, J., AND NI, L. M. A distributed scalable web server and its program visualization in multiple platforms. In *IEEE International Conference on Distributed Computing System Conference* (1996), pp. 665–672.

[74] LUO, Q., AND NAUGHTON, J. F. Form-based proxy caching for database-backed web sites. In *International Conference on Very Large Data Bases* (2001), pp. 191–200.

[75] LUO, Q., NAUGHTON, J. F., KRISHNAMURTHY, R., CAO, P., AND LI, Y. Active query caching for database Web servers. In *WebDB* (2000), pp. 92–104.

[76] MAH, B. A. An empirical model of http network traffic. In *IEEE Conference on Computer Communications* (1997), p. 592.

[77] MECELLA, M., SCANNAPIECO, M., VIRGILLITO, A., BALDONI, R., CATARCI, T., AND BATINI, C. Managing data quality in cooperative information systems. In *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002* (2002), pp. 486–502.

[78] MENASCÉ, D. A. Testing e-commerce site scalability with tpc-w. In *International Computer Measurement Group Conference* (2001), pp. 457–466.

[79] MOBASHER, B. Data mining for web personalization. In *The Adaptive Web* (2007), pp. 90–135.

[80] NAUMANN, F., FREYTAG, J. C., AND LESER, U. Completeness of information sources. In *Intl. Workshop on Data Quality in Cooperative Information Systems* (2003).

[81] NAUMANN, F., LESER, U., AND FREYTAG, J. C. Quality-driven integration of heterogenous information systems. In *International Conference on Very Large Data Bases* (1999), pp. 447–458.

[82] OKE, A., AND BUNT, R. B. Hierarchical workload characterization for a busy web server. In *International Conference on Objects, Models, Components and Patterns* (2002), pp. 309–328.

[83] OLSHEFSKI, D. P., NIEH, J., AND NAHUM, E. ksniffer: determining the remote client perceived response time from live packet streams. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation* (2004), USENIX Association, pp. 23–23.

[84] PADMANABHAN, V. N., AND QIU, L. The content and access dynamics of a busy web site: findings and implications. *SIGCOMM Comput. Commun. Rev. 30*, 4 (2000), 111–123.

[85] PADMANABHAN, V. N., AND QUI, L. The content and access dynamics of a busy web site: findings and implicatins. In *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (2000), pp. 111–123.

[86] PAI, V. S., DRUSCHEL, P., AND ZWAENEPOEL, W. Flash: An efficient and portable Web server. In *USENIX Symposium on Internet Technologies and Systems* (1999), pp. 199–212.

[87] PALLANT, J. *SPSS survival manual: A step by step guide to data analysis using SPSS for windows (version 10)*. Open University Press, 2001.

Stavros Papastavrou

[88] PAPASTAVROU, S., CHRYSANTHIS, P. K., SAMARAS, G., AND PITOURA, E. An evaluation of the java-based approaches to web database access. *International Journal of Cooperative Information Systems 10*, 4 (2001), 401–422.

[89] PAPASTAVROU, S., CHRYSANTHIS, P. K., SAMARAS, G., AND PITOURA, E. An evaluation of the java-based approaches to web database access. *Int. J. Cooperative Information Systems 10*, 4 (2001), 401–422.

[90] PAPASTAVROU, S., SAMARAS, G., EVRIPIDOU, P., AND CHRYSANTHIS, P. K. Fine-grained parallelism in dynamic web content generation: The parse dispatch and approach. In *CoopIS/DOA/ODBASE Conferences* (2003), pp. 573–588.

[91] PAPASTAVROU, S., SAMARAS, G., EVRIPIDOU, P., AND CHRYSANTHIS, P. K. Cfp taxonomy of the approaches for dynamic web content acceleration. 365–378.

[92] PAPASTAVROU, S., SAMARAS, G., EVRIPIDOU, P., AND CHRYSANTHIS, P. K. A decade of dynamic web content: A structured survey on past and present practices and future trends. *IEEE Communications Surveys & Tutorials 8*, 2 (2006), 52–60.

[93] PAULSON, L. D. Building rich web applications with ajax. *IEEE Computer 38*, 10 (2005), 14–17.

[94] PITOURA, E., CHRYSANTHIS, P. K., AND RAMAMRITHAM, K. Characterizing the temporal and semantic coherency of broadcast-based data dissemination. In *International Conference on Database Theory* (2003), pp. 407–421.

[95] POUR, G., AND XU, J. Javabeans, java, java servlets and cobra revolutionizing web-based enterprise application development. In *WebNet* (1999), pp. 895–900.

[96] PSOUNIS, K. Class-based delta-encoding: A scalable scheme for caching dynamic web content. In *IEEE ICDCS Workshops* (2002), pp. 799–805.

[97] QU, H., AND LABRINIDIS, A. Preference-aware query and update scheduling in web-databases. In *IEEE International Conference on Data Engineering* (2007), pp. 1–10.

[98] RABINOVICH, M., XIAO, Z., DOUGLIS, F., AND KALMANEK, C. R. Moving edge-side includes to the real edge - the clients. In *USENIX Symposium on Internet Technologies and Systems* (2003), pp. 87–95.

[99] RAMASWAMY, L., IYENGAR, A., LIU, L., AND DOUGLIS, F. Automatic fragment detection in dynamic web pages and its impact on caching. *IEEE Transactions on Knowledge and Data Engineering 17*, 6 (2005), 859–874.

[100] REDMAN, T. C. *Data Quality for the Information Age*. Artech House, Inc., 1997. Foreword By-A. Blanton Godfrey.

[101] RÖHM, U., BÖHM, K., SCHEK, H.-J., AND SCHULDT, H. Fas: a freshness-sensitive coordination middleware for a cluster of olap components. In *International Conference on Very Large Data Bases* (2002), pp. 754–765.

[102] SCHMITT, B., AND OBERLNDER, S. Access evaluation of digital libraries: Characteristics and performance of web opacs. In *Second Int. Workshop on New Developments in Digital Libraries* (2002).

Stavros Papastavrou

[103] SEGEV, A., AND FANG, W. Currency-based updates to distributed materialized views. In *IEEE International Conference on Data Engineering* (1990), pp. 512–520.

[104] SMITH, B., ACHARYA, A., YANG, T., AND ZHU, H. Exploiting result equivalence in caching dynamic web content. In *USENIX Symposium on Internet Technologies and Systems* (1999).

[105] SPRING, N. T., AND WETHERALL, D. A protocol-independent technique for eliminating redundant network traffic. In *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (2000), pp. 87–95.

[106] SRIVASTAVA, J., COOLEY, R., DESHPANDE, M., AND TAN, P.-N. Web usage mining: discovery and applications of usage patterns from web data. *SIGKDD Explor. Newsl. 1*, 2 (2000), 12–23.

[107] VEGLIS, A. A. Book review: Programming with coldfusion 5.0 (a review of programming coldfusion by rob brooks-bilson). *IEEE Distributed Systems Online 3*, 7 (2002).

[108] WANG, J. A survey of Web caching schemes for the Internet. *ACM Computer Communication Review 25*, 9 (1999), 36–46.

[109] WANG, Q., MAKAROFF, D., EDWARDS, H. K., AND THOMPSON, R. Workload characterization for an e-commerce web site. In *IBM Center for Advanced Studies International Conference* (2003), pp. 313–327.

[110] WANG, R. Y., AND STRONG, D. M. Beyond accuracy: what data quality means to data consumers. *J. Manage. Inf. Syst. 12*, 4 (1996), 5–33.

[111] WELSH, M., CULLER, D. E., AND BREWER, E. A. SEDA: An architecture for well-conditioned, scalable internet services. In *Symposium on Operating Systems Principles* (2001), pp. 230–243.

[112] WILSON, T. Review of gmail. *Information Research 10*, 1 (2004).

[113] WOLMAN, A., VOELKER, G. M., SHARMA, N., CARDWELL, N., KARLIN, A. R., AND LEVY, H. M. On the scale and performance of cooperative web proxy caching. In *Symposium on Operating Systems Principles* (1999), pp. 16–31.

[114] YAGOUB, K., FLORESCU, D., ISSARNY, V., AND VALDURIEZ, P. Caching strategies for data-intensive web sites. In *International Conference on Very Large Data Bases* (2000), pp. 188–199.

[115] YANG, C. C., AND WANG, F. L. Fractal summarization for mobile devices to access large documents on the web. In *International World Wide Web Conference* (2003), pp. 215–224.

[116] ZHANG, W., LIBERATORE, V., BEAVER, J., CHRYSANTHIS, P. K., AND PRUHS, K. Scalable data dissemination using hybrid methods. In *22nd IEEE Int'l Parallel and Distributed Processing Symposium* (2008).

Stavros Papastavrou